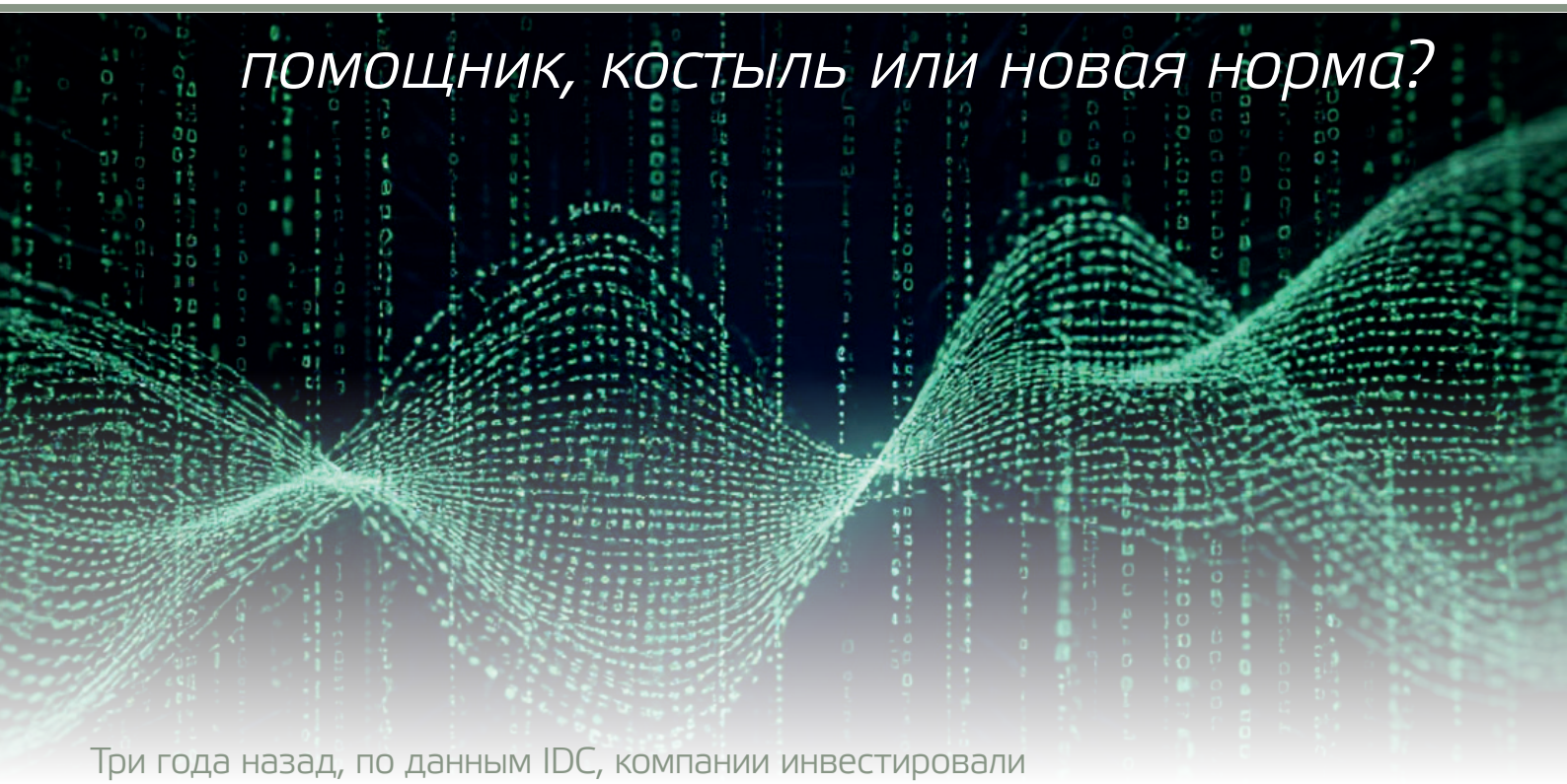
Нейросети в коде:

помощник, костыль или новая норма?



Три года назад, по данным IDC, компании инвестировали в развитие ИИ-технологий более 154 млрд долларов. К 2029 году эти инвестиции могут вырасти до 1,3 трлн долларов. Одна из основных статей расходов – новые инструменты для разработки. Искусственный интеллект постепенно интегрируется в рабочую среду. Мы попросили экспертов отечественной ИТ-отрасли поделиться своим мнением на сей счет.

1. *Что изменилось в работе разработчиков с распространением ИИ?*
2. *Сталкиваетесь ли вы в своих командах с ситуацией, когда разработчики генерируют с помощью ИИ не только код, но и тесты к нему?*
3. *Как часто ваши разработчики используют ИИ для создания архитектурных документов и проектных решений, которые выглядят убедительно на совещаниях, но не проходят глубокой инженерной проработки?*
4. *Традиционные навыки уступают месту умению «управлять промптами». Как это изменение повлияло на процесс онбординга junior-разработчиков в вашей компании и на ваши требования к их базовой инженерной подготовке?*
5. *Что может стать главной проблемой для ИТ-департаментов, когда ИИ перейдет из статуса «помощника, повышающего продуктивность», в статус основного автора кода в команде?*

На вопросы «БИТа» отвечают эксперты компаний

«Сегодня действует негласный закон выживания. Кто попросил машину написать функцию, тот никогда не доверяет ее проверкам. Тесты должен смотреть живой человек со стороны, иначе первый крупный релиз положит сайт на лопатки»

Владимир Нелюб,
директор по науке и ИИ, член правления ПАО «Группа Астра», управляющий партнер «Астра ИИ», д. т. н., профессор

Нейросети в программировании стали обычным турбонаддувом для любой ИТ-команды. Машина пока не едет сама по себе, хотя скорость разработки выросла пугающе быстро. Искусственный интеллект занял место стандартной программы на рабочем столе каждого разработчика. Он еще не уволил людей из профессии, зато полностью переписал их повседневные задачи.

Написание кода превратилось из чистого творчества в работу строгого корректора. Раньше программист часами искал примеры и писал каждую строчку вручную. Сегодня нейросеть выдает готовую заготовку за считанные секунды. Такой подарок несет скрытую угрозу. Время на проверку этой мгновенно созданной заготовки выросло пропорционально ее скорости. Разработчик теперь тратит основную часть смены на аудит чужого текста от машины. Ему нужно найти скрытые ошибки внутри очень уверенного в своей правоте алгоритма.

С тестами получается весьма коварный сюжет. Если попросить модель проверить ею же написанный код, она выдаст идеальный отчет для галочки перед руководством. Все показатели горят зеленым цветом. В реальности такая проверка часто оказывается обманом зрения чистой воды. Модель проверяет только самый простой путь пользователя.

Она не догадается проверить сбой при отключении интернета прямо во время оплаты картой или пять одновременных нажатий одной кнопки. Поэтому сегодня действует негласный закон выживания. Кто попросил машину написать функцию, тот никогда не доверяет ее проверкам. Тесты должен смотреть живой человек со стороны, иначе первый крупный релиз положит сайт на лопатки.

Проектирование новых систем выглядит как красивая картинка в глянцевом каталоге. Нейросеть рисует безупречные схемы и пишет весомые документы для начальства. Выглядит это крайне убедительно на совещаниях. На деле машина предлагает решения с покупкой десяти новых серверов. Ей неважен ограниченный бюджет отдела или возможности старого оборудования дата-центра. Человеку приходится брать эту красивую презентацию и долго приземлять ее на реальность существующих схем.

Сегодня нейросеть выдает готовую заготовку за считанные секунды. Такой подарок несет скрытую угрозу

Главный актив организаций будущего – живые люди со здравым смыслом. Именно они должны оставлять за собой право последнего слова

Раньше джуниора мучили вопросами про сложные математические алгоритмы. Сегодня знание формул отошло далеко на задний план. Главное требование к молодому специалисту стало звучать иначе. Джун обязан обладать развитым чутьем на халтуру. Он должен прочитать сто строк машинного кода и сразу понять скрытую угрозу. Здесь база данных захлебнется под нагрузкой пользователей. А вот здесь хитрый хакер найдет лазейку для кражи личных данных клиентов.

Стажер, умеющий только правильно спрашивать чат-бота, сегодня абсолютно бесполезен компаниям. Нужны люди, понимающие механику системы достаточно хорошо для своевременной остановки слишком умной программы.

Главная проблема будущего выглядит тревожно. Когда большая часть кода будет написана машиной, команды потеряют понимание собственных продуктов. Никто толком не сможет объяснить причины текущей внутренней архитектуры. При аварии инженеры посмотрят на тысячи строк чужого текста и побоятся вносить исправления руками. Они начнут запускать новые промпты, наслаивая старые проблемы друг на друга словно плохой ремонт. Кроме того, обученная на старом коде машина всегда станет предлагать решения из прошлого. Она надежно заблокирует появление чего-то действительно нового своей статистической уверенностью.

В этих условиях успех ждет совсем другие команды. Побеждать будут вовсе не те компании, которые первыми откажутся от живых разработчиков ради красивых цифр автоматизации. Главный актив организаций будущего – это живые люди со здравым смыслом. Именно они должны оставлять за собой право последнего слова. Только человек способен вовремя сказать цифровому помощнику «нет» ради сохранения стабильности сервиса. За этим отказом стоят реальные деньги бизнеса и спокойствие миллионов обычных пользователей». БИТ

«Помощник, костыль или норма – вопрос сводится к одному: ИИ становится нормой ровно в той мере, в какой инженеры продолжают понимать, что происходит внутри решения, которое они подписывают своим именем»

Александр Бочкин,
генеральный директор Инфомаксимум

1. С распространением ИИ меняется сама ценность разработчика. Парадоксально прозвучит, но владение конкретным языком программирования уже не главный навык для специалиста. Сейчас важнее умение понять бизнес-задачу, спроектировать архитектуру решения, продумать логику продукта/ограничения/потенциальные риски.

ИИ – это руки: он формирует типовые фрагменты кода, помогает собрать прототип, предлагает варианты реализации и готовит документацию.

2. Да, и именно генерация тестов стала одним из первых сценариев, которые разработчики начали массово передавать искусственному интеллекту. Многие программисты даже честно говорят – им нравится писать код, но вот делать все остальное вокруг этого, нет. Формирование основной логики часто занимает меньше времени, чем подготовка тестов, документации и проверка различных вариантов поведения системы. Это рутинная работа, с которой нейросети справляются особенно эффективно.

Чем проще генерация кода, тем важнее оценить качество результата, увидеть архитектурную ошибку, учесть нагрузку, безопасность и быть в курсе развития системы

За счет автоматизации этих операций заметно сокращается время от постановки задачи до готового решения. Качественный рывок в возможностях нейросетей в конце 2025 года усилил этот эффект: простые автоматизации теперь создаются за 30–60 минут, сложные – за несколько дней.

Одновременно стираются границы между отдельными специализациями. Разработчик может работать с более широким набором задач, даже если раньше был сосредоточен только на backend- или frontend. Это не означает, что глубокая техническая экспертиза больше не нужна. Тут дело в другом: чем проще генерация кода, тем важнее способность оценить качество результата, увидеть архитектурную ошибку, учесть нагрузку, безопасность и быть в курсе дальнейшего развития системы.

Разработчик все чаще выступает инженером, отвечающим за конкретный продукт или функциональную область. ИИ в этой модели – инструмент, который ускоряет реализацию.

Разработчик может передать модели фрагмент кода или описание функции и получить набор unit-тестов, граничных сценариев и вариантов входных данных. ИИ также помогает проверить, какие условия были пропущены и какие ошибки могут возникнуть при нестандартном поведении пользователя или системы.

При этом автоматически сгенерированные тесты нельзя принимать без проверки. Модель работает в рамках переданного ей контекста и может не учитывать реальные зависимости, особенности инфраструктуры или бизнес-логику продукта. Поэтому задача разработчика – определить, что именно необходимо проверить, оценить полноту покрытия и убедиться, что тесты действительно отражают требования. ИИ ускоряет эту работу, но ответственность за качество кода и его поведение в промышленной среде остается на команде.

С архитектурными документами работает похожая ловушка, и на совещании она особенно заметна на слух. Модель формулирует любое решение с одинаковой уверенностью: и то, что продумано на несколько ходов вперёд, и то, что собрано за пять минут ради красивого слайда. Убедительность такого текста – это гладкость формулировок, а глубина проработки может отсутствовать вовсе. Разница становится видна, когда кто-то в комнате задаёт простой вопрос: что случится при отказе этого узла, почему выбран именно такой подход. Документ без инженера за штурвалом такую проверку проходит редко – за гладким текстом обнаруживается пустота.

Большого штата и потока джунов у меня нет, поэтому об онбординге сужу со стороны – по проектам, где разворачиваю клиентам рабочие ИИ-терминалы и настраиваю их под конкретные задачи команды. Там у новичка важнее умение читать чужой вывод и находить в нём слабое место, чем помнить синтаксис языка наизусть. Базовая инженерная подготовка от этого становится ценнее: именно она даёт возможность отличить рабочее решение от текста, который лишь выглядит рабочим.

Когда работа с ИИ станет для инженерных команд обычной практикой, главная проблема сместится с качества кода на способность объяснить, почему система устроена именно так. Решение сегодня часто рождается за несколько минут диалога с моделью, и через полгода авторство этой логики принадлежит истории переписки с ней гораздо больше, чем конкретному человеку в команде. Когда система ломается, инженеру приходится восстанавливать замысел заново – как будто он читает код коллеги без единого комментария, хотя формально сам его писал. Способность объяснить систему становится редким навыком именно тогда, когда она нужнее всего.

Помощник, костыль или норма – вопрос сводится к одному: ИИ становится нормой ровно в той мере, в какой инженеры продолжают понимать, что происходит внутри решения, которое они подписывают своим именем. БИТ

«Архитектурный документ, созданный ИИ, я воспринимаю только как черновик для инженерной дискуссии, а не как готовое решение»

Евгений Фроликов,
Java-архитектор, эксперт по безопасности, основатель BitDive

ИИ уже заметно изменил саму модель работы разработчика. Раньше большая часть времени уходила на написание кода, поиск примеров, чтение документации и разбор типовых ошибок. Сейчас значительную часть этой работы можно делегировать ИИ. В результате хороший разработчик пишет меньше кода вручную, но должен гораздо больше времени тратить на проверку того, что было сгенерировано. И здесь возникает главный риск: скорость создания кода выросла быстрее, чем способность команд его качественно проверять.

Мы уже видим ситуацию, когда разработчик с помощью ИИ генерирует не только реализацию, но и тесты к ней. Формально всё выглядит отлично: есть код, есть тесты, CI зелёный. Но если и реализацию, и проверку написал один и тот же ИИ на основании одного контекста, тесты очень часто подтверждают логику самого решения, а не реальные требования бизнеса. Получается своеобразная «самопроверка». Поэтому я считаю, что ценность будут получать системы, которые способны проверять AI-код независимо – например, через реальные runtime-сценарии, интеграционные вызовы и фактическое поведение приложения.

Похожая проблема есть и с архитектурой. ИИ очень хорошо умеет создавать убедительные документы: красивые диаграммы, правильные термины, Kafka, Redis, CQRS, event-driven architecture. На совещании такое решение может выглядеть профессионально. Но при глубоком разборе иногда оказывается, что не просчитаны отказоустойчивость, консистентность, эксплуатационная стоимость, миграция данных или реальные нагрузки. Поэтому архитектурный документ, созданный ИИ, я воспринимаю только как черновик для инженерной дискуссии, а не как готовое решение.

Особенно сильно это влияет на junior-разработчиков. Раньше человек был вынужден самостоятельно пройти через десятки ошибок и благодаря этому постепенно понимал, как работает код. Сейчас можно получить рабочее решение за несколько минут, практически не понимая его устройства. Поэтому требования к фундаментальной подготовке, на мой взгляд, не должны снижаться – наоборот. Нам всё меньше нужен человек, который просто умеет писать синтаксически корректный Java-код, и всё больше нужен инженер, который понимает транзакции, конкурентность, базы данных, сети, безопасность и способен объяснить, почему сгенерированное решение действительно работает.

Следующим этапом развития AI-разработки, на мой взгляд, станет не ещё более быстрое написание кода, а независимая автоматическая проверка его реального поведения

Главная проблема наступит тогда, когда ИИ станет основным автором кода, а человек – только его оператором. Компании начнут накапливать огромный объём программного обеспечения, происхождение и поведение которого никто до конца не понимает. Поэтому следующим этапом развития AI-разработки, на мой взгляд, станет не ещё более быстрое написание кода, а независимая автоматическая проверка его реального поведения. ИИ как автор кода действительно станет нормой. Но инженерная ответственность за результат всё равно останется на человеке. БИТ

«Если и код, и тест к нему написал один и тот же алгоритм по одной и той же логике, то тест с высокой вероятностью просто подтвердит ошибку кода, а не поймает ее»

Александр Мезенцев,
CIO финтех-компании MoneyCat , автор ТГ-канала «АйТи мой хороший»

1. Раньше день разработчика в основном уходил на то, чтобы превратить задачу в работающий код: вспомнить синтаксис, найти нужный метод, набросать шаблон, отладить очевидные ошибки. Сейчас это все за секунды делает ассистент. И тут дело не столько в самом коде, сколько в том, куда смещается внимание человека. Рутина уходит, и это хорошо, но вместе с ней уходит и «мышечная память», которая как раз на этой рутине и нарабатывалась годами. Разработчик все чаще не пишет решение сам, а только оценивает, что предложил алгоритм и решает, пускать в прод или нет. А это, чего греха таить, совсем другой навык, и не факт, что он в конечном итоге принесет столько же пользы, сколько старый.

2. Тесты, сгенерированные вместе с кодом – это давно норма, и она сама по себе ускоряет разработку, тут спорить не с чем. Проблема в том, что если и код, и тест к нему написал один и тот же алгоритм

по одной и той же логике, то тест с высокой вероятностью просто подтвердит ошибку кода, а не поймает ее. По моим наблюдениям, именно это сейчас редко замечают на уровне рядовых команд: покрытие тестами растет, метрики, конечно, радуют глаз, но реальная защита от багов за ним не поспевает.

Мы рискуем получить поколение специалистов, которые отлично «умеют в промпты», но не понимают, чем управляют, это станет заметно не сразу, а только когда что-то пойдет не так в системе

3. ИИ умеет отлично создавать структурированный, красиво звучащий текст со всеми разделами, пояснениями, если нужно, нарисует нужные схемы-диаграммы. Конечно, на совещании такой документ выглядит убедительнее, чем сырые заметки человека, который реально потратил время на продумывание архитектуры, а потому не успел красиво все оформить. Но нужно понимать, что красивая картинка и настоящая работа инженера – это разные вещи, и вторая проверяется не на совещании, а через полгода в проде, когда систему заштормит в тех ситуациях, которые в презентации даже не были упомянуты.

4. Здесь рынок, на мой взгляд, идет по опасной траектории. Соблазн упростить требования к джунам понятен, ведь раз ИИ уже закрывает рутинный код, пусть просто учатся формулировать задачи для модели. Но на самом деле, это может потом сильно аукнуться. Промпт-инжиниринг без инженерной базы дает человека, который может получать от ИИ код, с виду рабочий, но такой спец не способен объяснить, да возможно и понять, почему решение сломается в конкретной ситуации. Так что требования к джунам должны не снижаться, а наоборот повышаться и смещаться в сторону большего системного мышления. А вот с этим навыком сейчас как раз дефицит.

5. Страшно представить, что будут уверенно делать алгоритмы через пару-тройку лет, если уже сейчас они закрывают рутинный код, участвуют в тестировании и пишут простые микросервисы. ИИ одновременно сжимает время входа для молодых кадров в профессию и поднимает планку этого входа. Работодателю больше не нужен дорогой джун-исполнитель, нужен именно постановщик задач для агента. По сути, каждый разработчик постепенно превращается в руководителя для своего исполнителя в лице нейросети.

И здесь кроется главный риск для ИТ: если этот, так сказать, «руководитель» сам никогда не проходил через боль отладки, через понимание, почему решение ломается, он не сможет отличить выданный ему готовый результат агента от возможных ошибок. Мы рискуем получить поколение специалистов, которые отлично «умеют в промпты», но совершенно не понимают, чем управляют, это станет заметно не сразу, а только в тот момент, когда что-то пойдет не так в системе, а в компании уже никто не сможет объяснить проблему на уровне кода, разве что на уровне «она вроде бы должна делать то-то и то-то». БИТ

«ИИ не отменяет разработчиков. Он довольно быстро отменяет часть их рутины. И, на мой взгляд, это две совершенно разные вещи»

Николай Симонов,
коммерческий директор ООО «Андагар»

Сегодня разработчик может за минуты получить код, тесты или несколько вариантов реализации задачи. Но сгенерировать – не значит решить. ИИ не отвечает за то, насколько это решение вписывается в архитектуру проекта, выдержит ли реальную нагрузку, безопасно ли оно с точки зрения данных и не создаст ли технический долг через полгода.

Мы в компании действительно иногда используем ИИ для генерации кода, тестов и анализа. Но здесь есть важный момент: результат ИИ – это не финальный ответ, а материал для работы специалиста. Если разработчик не способен проверить то, что ему предложила модель, то проблема не в ИИ.

Поэтому я бы не соглашался с тезисом, что традиционные навыки разработчика теперь уступают место умению писать промпты. Умение работать с ИИ становится базовым навыком, но хороший промпт не заменяет понимание архитектуры, алгоритмов, принципов разработки и тестирования. Более того, чем активнее мы используем ИИ, тем выше требования к способности человека критически оценивать результат.

Главный риск для ИТ-департаментов начнется тогда, когда ИИ станет восприниматься как автор, которому можно без проверки передать ответственность за код. Тогда очень легко получить много быстро написанного кода и очень мало качественного результата. А переделка займет еще больше времени и денег. Хотя и это скорее вопрос времени: с текущей скоростью развития ИИ, уже через 1-2 года, наверняка, и код ИИ будет писать лучше, и доверия к нему может быть больше.

Я бы вообще не ставил вопрос «ИИ или разработчик». Хороший специалист с ИИ гораздо сильнее, чем хороший специалист без него. ИИ берет на себя простые и, как уже говорил выше, рутинные задачи, ускоряет процесс, а человек задает направление, принимает решения и отвечает за результат. На мой взгляд, именно это сочетание уже стало новой нормой в разработке. БИТ

«В интеграционных проектах ИИ позволяет за час-два набросать каркас адаптера и проверить гипотезу, что делает оценку полноценной интеграции более точной по трудоемкости и длительности»

Наталия Ефимцева,
системный архитектор ICL Services

1. С распространением ИИ я вижу в своих проектах следующие изменения:

- Сдвиг фокуса с «написания кода» на «проектирование и контроль». Разработчики тратят меньше времени на механическое написание шаблонного кода и больше на архитектуру, логику, интеграции, безопасность и принятие решений. ИИ легко может написать код (много кода) под любой запрос, насколько этот код будет правильным, безопасным, удобным в дальнейшей эксплуатации и поддержке во многом зависит от «промпта». Конечно, никто не пишет продуктивный код одним промптом, промпты должны начинаться с промтов для проектирования того или иного функционала, правильно заданных ограничений, описание модулей и их взаимодействия. И только после «согласованного» регламента последует промпт для написания кода.
- Скорость написания кода перестаёт быть главным KPI. На первый план выходят умение проектировать, оценивать риски, держать архитектурную целостность, понимать экономику решения и нести ответственность за итоговый результат.
- Кратное ускорение прототипирования. В интеграционных проектах ИИ позволяет за час-два набросать каркас адаптера и проверить гипотезу, что делает оценку полноценной интеграции более точной по трудоемкости и длительности. У нас уже было несколько проектов, в которых на этапе пресейла проверяли гипотезы (и эмулировали интерфейс) и выбирали наиболее подходящую по технической реализации и бизнес функционалу. Ранее на этапе пресейла такая детализация была невозможна из-за трудоемкости и длительности.

4. ИИ – это компилятор высокого уровня. Когда-то мы писали на ассемблере (ручной код), теперь пишем на псевдокоде (промпты). «Умение управлять промптами» не отменяет фундамент, а надстраивается над ним. Базовая инженерная подготовка остаётся обязательной. Без понимания основ алгоритмов и структур данных, принципов работы ОС/сетей/баз данных, основ тестирования, отладки и работы с Git специалист (разработчик/DevOps) не сможет адекватно оценивать и исправлять код, сгенерированный ИИ.

Промпт-инжиниринг становится дополнительным базовым навыком: умение формулировать задачу для ИИ (контекст, ограничения и т.п.), навыки итеративного уточнения промптов, понимание, где ИИ надёжен, а где склонен «галлюцинировать».

Вместо проверки знания синтаксиса мы оцениваем способность выстраивать причинно-следственные связи между компонентами системы. Задача онбординга – не научить пользоваться ИИ, а научить декомпозировать бизнес-задачу до уровня атомарных модулей, чтобы генерация кода становилась лишь техническим шагом, учим формулировать запросы к ИИ через архитектурные ограничения (таймауты, ретраи, идемпотентность), а не через «сделай как-нибудь». БИТ

Вместо проверки знания синтаксиса мы оцениваем способность выстраивать причинно-следственные связи между компонентами системы

«ИИ может быть автором кода, но не должен становиться владельцем инженерного решения. Ответственность по-прежнему должна оставаться у человека»

Сергей Симонов,
заместитель декана факультета «Информационные технологии» МТУСИ

На мой взгляд, мы уже прошли этап, когда использование нейросетей в программировании можно было рассматривать просто как дополнительный инструмент разработчика. ИИ постепенно становится частью стандартного инженерного процесса. При этом принципиально меняется не столько скорость набора кода, сколько сама структура работы программиста: все больше времени смещается от непосредственного написания кода к постановке задачи, анализу предложенного решения, проверке, тестированию и интеграции результата в существующую систему. Фактически разработчик постепенно становится не только автором, но и редактором, верификатором и архитектором машинно-сгенерированного решения.

В наших проектах мы, конечно, сталкиваемся с использованием ИИ и для генерации кода, и для подготовки тестов. Само по себе это не является проблемой. Напротив, генерация типовых unit-

тестов, тестовых данных, документации или шаблонного кода – вполне рациональный способ сократить рутинную работу. Риск возникает тогда, когда один и тот же ИИ фактически выступает и автором решения, и автором проверки этого решения. Тесты могут воспроизводить те же ошибочные предположения, которые были заложены в сгенерированный код. Поэтому наличие автоматически созданных тестов еще не означает, что решение действительно верифицировано.

Похожая проблема возникает с архитектурными документами. Современные модели очень хорошо умеют создавать тексты, которые внешне выглядят профессионально: правильная терминология, диаграммы, описание компонентов, аргументация выбора технологий. На совещании такой документ может производить впечатление глубоко проработанного решения. Но архитектура проверяется не качеством презентации, а тем, насколько автор понимает ограничения системы: нагрузку, отказоустойчивость, безопасность, стоимость эксплуатации, модель данных, интеграционные зависимости и последствия принимаемых компромиссов. Поэтому для меня один из новых профессиональных рисков – возникновение иллюзии инженерной глубины, когда качество формы начинает опережать качество содержания.

Ошибки были и в коде, написанном человеком. Более серьезный риск – постепенная утрата командой способности самостоятельно понимать создаваемую систему

Особенно важен этот вопрос применительно к junior-разработчикам. Я бы не сказал, что традиционные навыки уступают место «умению писать промпты». Скорее происходит обратное: по мере роста возможностей ИИ фундаментальная подготовка становится еще важнее. Опытный специалист способен быстро увидеть, где модель предлагает сомнительное решение. Начинающий разработчик зачастую еще не обладает критериями, позволяющими отличить хороший код от просто убедительно выглядящего. Поэтому при обучении и онбординге необходимо проверять не способность получить от нейросети работающий фрагмент программы, а понимание алгоритмов, структур данных, архитектурных принципов, баз данных, сетевого взаимодействия, тестирования, информационной безопасности. Важным становится умение ответить на вопрос: почему это решение работает и при каких условиях оно перестанет работать?

С точки зрения руководителя ИТ-команды, главная долгосрочная проблема заключается даже не в возможных ошибках ИИ. Ошибки были и в коде, написанном человеком. Более серьезный риск – постепенная утрата командой способности самостоятельно понимать создаваемую систему. Если ИИ становится основным автором кода, а разработчики ограничиваются постановкой задач и поверхностной проверкой результата, возникает своего рода инженерный долг – по аналогии с техническим долгом. Компания получает все больший объем программного кода, за который формально отвечает команда, но который она понимает все хуже.

Поэтому я бы сформулировал новую норму следующим образом: ИИ может быть автором кода, но не должен становиться владельцем инженерного решения. Ответственность за архитектуру, критерии качества, безопасность, тестирование и понимание последствий по-прежнему должна оставаться у человека. В этом смысле ценность сильного разработчика в эпоху ИИ не уменьшается. Меняется предмет его профессионального мастерства: от способности быстро написать много кода – к способности поставить правильную задачу, выбрать решение, проверить его и нести за него инженерную ответственность. БИТ

«Отследить использование ИИ без установки программ мониторинга на рабочие станции невозможно. Российский рынок сейчас абсолютно не защищен: внешние поставщики теоретически могут внедрить вредоносный код для пользователей из определенных регионов»

Роман Садрисламов,
директор производственного направления «Девелоники» (fabricaONE.AI, акционер – ГК Softline)

С внедрением искусственного интеллекта в разработку сформировались два сценария использования этих инструментов. Первый – стихийное применение. Большая часть компаний не знает, как их разработчики используют системы искусственного интеллекта. Специалисты сами покупают подписки и применяют их для ускорения работы, высвобождая личное время. Фактическая продолжительность работы может сокращаться вдвое, но руководство об этом не догадывается. При этом корпоративный код передается внешним системам без контроля, что создает серьезные бреши в информационной безопасности.

Второй сценарий – целенаправленное внедрение. Некоторые компании официально внедряют ИИ-инструменты и одновременно повышают показатели производительности для разработчиков. Вообще, на ИТ-рынке все чаще слышно, что применение интеллектуальных систем включается в обязательные ус-

В нашей компании за полгода мы разработали собственную платформу и более 10 ИИ-агентов, которые уже вошли в ежедневную рабочую жизнь производственных сотрудников

ловия работы. Компании компенсируют стоимость подписок или включают эти расходы в зарплату. И уже без новых инструментов специалисты просто не могут выполнить поставленные задачи.

В нашей компании за полгода мы разработали собственную платформу и более 10 ИИ-агентов, которые уже вошли в ежедневную рабочую жизнь производственных сотрудников.

Объективно разработчики с ИИ тратят меньше времени на задачи, и главная проблема в том, что сейчас нет эффективных методов контроля со стороны бизнеса. Отследить использование ИИ без установки программ мониторинга на рабочие станции невозможно. Российский рынок сейчас абсолютно не защищен: внешние поставщики теоретически могут внедрить вредоносный код для пользователей из определенных регионов.

Генерация автоматизированных тестов стала базовой практикой. Именно через тесты мы контролируем качество кода, который создают интеллектуальные системы. Раньше разработчики редко писали автоматизированные тесты – модульные, интеграционные: это требовало времени. Проще было отдать продукт тестировщику. Теперь генерация тестов почти ничего не стоит по времени.

Изменилось распределение функций: объем работы разработчика вырос за счет автоматизированных тестов, а тестировщики получили разгрузку. Ручное тестирование не исчезло, но баланс сместился. В результате общее качество кода выросло благодаря автоматизации проверок.

В серьезной архитектурной проработке искусственный интеллект применяется ограниченно и только как вспомогательный инструмент. Мы используем специальный ИИ-агент для архитекторов, разработанный по их требованиям. Он помогает создавать схемы, но ключевое слово – «помогает». Финальные решения, особенно по верхним уровням архитектуры и критическим компонентам, принимает только человек. Поручать ИИ самостоятельное проектирование сложных систем рано. Архитектурные решения требуют понимания бизнес-контекста, технических ограничений, перспектив развития – это пока за пределами возможностей интеллектуальных систем.

В серьезной архитектурной проработке искусственный интеллект применяется ограниченно и только как вспомогательный инструмент

Традиционные навыки уступают место умению «управлять запросами. Четких показателей нет, но ожидания изменились. Мы не принуждаем новых сотрудников использовать ИИ-инструменты, но поощряем инициативу. Период адаптации стараемся сокращать за счет использования интеллектуальных систем, но полностью исключить его невозможно.

Принципиальный момент: базовая инженерная подготовка остается критически важной. Начинающий специалист должен понимать принципы разработки, архитектуры систем, алгоритмов. Умение формулировать запросы к интеллектуальным системам – это надстройка над базовыми знаниями, а не их замена. Специалист, который не понимает, что запрашивает и как оценить результат, не сможет работать эффективно даже с лучшими инструментами.

Что может стать главной проблемой для ИТ-департаментов, когда ИИ перейдет из статуса «помощника, повышающего продуктивность», в статус основного автора кода в команде? Этот переход мы наблюдаем прямо сейчас, и проблемы становятся очевидными.

Первая проблема – утрата контроля. Уже сейчас есть специалисты, которые работают так: человек формулирует требования, набор автоматизированных инструментов пишет код, развертывает, проверяет, исправляет ошибки, повторяет цикл. При этом такой специалист не всегда может ответить, на каком языке программирования реализована система.

Еще одна проблема – масштаб кодовой базы. Со временем объем кода разрастается. Когда система содержит сотни тысяч строк автоматически сгенерированного кода, проверить корректность каждого фрагмента невозможно. Управляемость теряется, предсказуемость поведения снижается. Код может быть идеальным, а может содержать критические ошибки – «бомбы замедленного действия», которые сработают в неподходящий момент.

Третья проблема – информационная безопасность. Злоумышленники активно сканируют открытые хранилища кода, выявляют уязвимости и взламывают производственные системы. Интеллектуальные системы при написании кода могут использовать библиотеки, которые функционально подходят, но содержат известные уязвимости. Система не может проводить анализ безопасности каждой библиотеки – это слишком затратно. Справедливости ради, аналогичные ошибки совершает и человек, но масштаб и скорость генерации кода интеллектуальными системами многократно увеличивают риски.

Ключевой вызов для ИТ-департаментов – контролировать качество и безопасность автоматически генерируемого кода, сохраняя преимущества в скорости разработки. БИТ

«Умение правильно поставить задачу нейросети полезно, но оно не заменяет знания алгоритмов, баз данных, сетевого взаимодействия, безопасности и тестирования»

Вячеслав Коэн,
владелец агентств Usertech и Time-to

Нейросети уже стали обычным рабочим инструментом для разработчика. Они помогают писать типовые функции, SQL-запросы и тесты, быстрее осваивать новые фреймворки, искать ошибки, разбираться в чужом коде и готовить документацию. За счёт этого первый жизнеспособный вариант решения задачи появляется быстрее. Но в реальном проекте недостаточно получить работающий код. Нужно понимать, как он связан с другими частями системы, соответствует ли бизнес-задаче и не создаст ли проблем при дальнейшем развитии продукта.

В нашей основной практике – разработке сайтов – ИИ лучше всего справляется с небольшими задачами, результат которых легко проверить. Например, можно поручить ему написать регулярное выражение, переработать отдельную функцию или предложить варианты исправления ошибки. Чем сложнее задача и чем больше в ней внутренней логики проекта, тем выше риск получить решение, которое выглядит правильным только на первый взгляд. Код может работать, но при этом создавать лишнюю нагрузку на базу данных, нарушать принятые правила разработки или неправильно обрабатывать редкие ситуации. Поэтому сгенерированный код должен проходить обычное ревью. Разработчик обязан понимать, что именно он добавляет в проект и почему выбран такой подход.

ИИ активно используют и для написания тестов. Это удобно, ведь он быстро подбирает типовые сценарии, входные данные и граничные значения. Однако нельзя поручать нейросети одновременно написать код и полностью проверить его. Если модель изначально неправильно поняла задачу, она может повторить ту же ошибку в тестах. Все проверки будут успешно проходить, хотя сама функция работает не так, как требуется бизнесу. Поэтому сначала нужно отдельно определить ожидаемый результат, а затем проверять, действительно ли тесты находят ошибки. Простой способ – намеренно сломать условие в коде и посмотреть, заметит ли это тест.

Похожая проблема возникает с архитектурными документами. Нейросеть умеет составлять убедительные описания, добавлять схемы, микросервисы, очереди и кэширование. На совещании такое решение может выглядеть профессионально, но не учитывать реальную нагрузку, стоимость инфраструктуры, безопасность или перенос данных из старой системы. Мы оцениваем подобные документы не по количеству терминов, а по конкретике. Автор должен объяснить, почему выбран именно этот вариант, какие решения рассматривались ещё, сколько это будет стоить и что произойдёт при сбое. Если без помощи ИИ он не может ответить на эти вопросы, значит, проект ещё не проработан.

Изменился и подход к junior-разработчикам. Умение правильно поставить задачу нейросети полезно, но оно не заменяет знания алгоритмов, баз данных, сетевого взаимодействия, безопасности и тестирования. Более того, начинающий специалист теперь может за несколько минут создать большой объём кода, который пока не способен полноценно проверить. Поэтому при онбординге мы смотрим не на скорость написания кода, а на то, может ли человек объяснить решение, найти слабые места и самостоятельно исправить ошибку.

Главный риск появится тогда, когда ИИ превращается из помощника в основного автора кода. В таких случаях кодовая база начинает расти быстрее, чем команда успевает разобраться в её устройстве. В результате любое изменение требует больше времени, а риск ошибок и уязвимостей увеличивается. Поэтому нейросети уже стали частью разработки, но использовать их стоит как инструмент, а не как замену инженерной экспертизе. БИТ

«ИИ в разработке это уже новая норма, с которой отрасль ещё не научилась работать»

Дмитрий Фырнин,
управляющий партнёр и технический директор SENSE

Главное изменение – рутинные операции стали быстрее. Но там, где принято писать ИИ-код, растёт технический долг: если кода много, то следить за его качеством проблематично. Особенно если тесты потом тоже пишет агент. Процессы сейчас пересматриваются, сообщество экспериментирует со spec-driven development, но гайда в духе «делай так и будет счастье» пока нет. Отраслевого стандарта тоже.

Ещё одно следствие – многих сеньоров стал раздражать ИИ-код от джунов и мидлов-минус. На этом уровне сложно понять архитектурно, когда задачу можно отдать нейросети, а когда код нужно собственноручно вычищать. Джуны часто не развиваются в архитектуре и как раз архитектурные вопросы отдают ИИ, а компетенции проконтролировать результат не хватает. Отсюда праведное возмущение коллег: зачем платить полную зарплату джуну, который всё делает через ИИшку.

Про тесты. Тесты нейросетью тоже можно генерировать, и здесь важен порядок действий. Если код уже написан, и вы просите ИИ покрыть его тестами, он покроет, но у нейросети есть проблемы с граничными случаями, а хороший инженер контроля качества в первую очередь ими и занимается. Обратный порядок работает лучше: QA генерирует тесты по спецификации, полирует их, просматривает граничные случаи, и только потом задача уходит в разработку, в том числе агенту, которому запрещено эти тесты менять.

Приживётся такой подход или нет – посмотрим.

Про архитектурные документы. Убедительные на совещании, но не проработанные инженерно архитектурные документы – история известная, такие случаи бывают. При этом сам по себе ИИ как консультант по архитектуре – скорее добро. Для большинства задач человечество уже выработало эффективные подходы, и прежде, чем изобретать велосипед, стоит изучить придуманное. Проблема в том, что нейросеть не использует инженерный подход, который начинается с вопроса, а нужна ли вообще эта фича. Нейросеть не скажет: «не делай это, лучше оптимизируй процесс и фича не понадобится». Просят – делает.

Про джунов и промптинг. С тезисом о том, что традиционные навыки уступают место умению управлять промптами, я не соглашусь. Это верно, только если нанимать такого, пока выдуманного сотрудника, как оператор LLM. При найме разработчика, тем более джуна, наша задача – оценить инженерное мышление, и требования к базовой подготовке мы не снижали.

При найме разработчика, тем более джуна, наша задача – оценить инженерное мышление, и требования к базовой подготовке мы не снижали

Между джуном с хорошей инженерной подготовкой и человеком, который невероятно круто пишет промпты, я не задумываясь выберу инженера: он способен оценить результат.

Что будет, когда ИИ станет основным автором кода? ИТ-департаменты столкнутся с двумя проблемами.

Первая – разрастание техдолга. Чтобы он не нарастал, весь код должен проходить подробное код-ревью, а на большом проекте нужен ещё и архитектурный комитет, который следит, чтобы на похожих задачах в разных местах использовались похожие решения. Чем быстрее растёт техдолг, тем дороже писать следующую строчку кода. В какой-то момент в нём начнёт захлёбываться сама нейросеть и будет присылать изменения на тысячу файлов, которые никто не сможет вычитать и внятно протестировать. Чтобы запустить такой коммит в продакшн, нужна будет вера. А вера и инженерное мышление – вещи противоречащие.

Вторая проблема – размытие ответственности. За код всегда кто-то должен отвечать, и этот кто-то – человек. Когда в три часа ночи в субботу падает прод, ответственный – разработчик. Но если основным автором кода мы называем нейросеть, какой смысл в этом назначении, если отвечает всё равно не она? БИТ

«Согласованность между кодом и тестом создаёт иллюзию надёжности – измеряет она лишь то, насколько модель последовательна сама с собой, а вопрос, правильно ли решена сама задача, остаётся открытым»

Тим Зинин,
управляющий партнер компании «Зинин, Штурбин и партнёры»

Разработчик, который работает в паре с моделью каждый день, тратит меньше времени на написание кода и больше времени на проверку того, что модель выдала. Она собирает рабочий фрагмент за секунды, и центр работы смещается с печатания на решение: стоит ли доверять этому фрагменту и правильно ли вообще сформулирована задача. Тест к этому же коду часто пишет та же модель – и это опаснее, чем кажется со стороны. Такой тест чаще воспроизводит логику уже написанного решения, чем сверяет её с реальным требованием: модель строит оба артефакта из одного понимания задачи, и, если это понимание ошибочно, тест подтверждает ошибку с той же уверенностью, с какой код её совершил. Согласованность между кодом и тестом создаёт иллюзию надёжности – измеряет она лишь то, насколько модель последовательна сама с собой, а вопрос, правильно ли решена сама задача, остаётся открытым. БИТ

«Наше правило: сценарий для критичной бизнес-логики формулирует человек до реализации. Процент покрытия при этом ничего не показывает: полезнее сломать код специально и проверить, заметят ли это тесты»

Антон Фокин,
CEO Qtim

1. Узкое место переехало с написания кода на его проверку. Кода появляется в разы больше, а скорость, с которой команда способна его принять, прежняя. Изменился и профиль ошибки: раньше она падала и была видна, теперь выглядит как аккуратный рабочий код, который делает не то – запрос правильной формы, но не к той таблице. При беглом ревью это не ловится. Так что ни помощник, ни костыль:

новая норма, в которой подорожала проверка.

2. Когда разработчики генерируют с помощью ИИ не только код, но и тесты к нему, это опаснее сгенерированного кода: код кто-то читает, зелёные тесты – никто. Модель пишет тест по коду, а не по требованию, и фиксирует фактическое поведение вместе с багом. Отсюда моки поверх той самой логики, ради которой тест писался, и assert под фактический результат. Наше правило: сценарий для критичной бизнес-логики формулирует человек до реализации. Процент покрытия при этом ничего не показывает: полезнее сломать код специально и проверить, заметят ли тесты.

3. Когда используют ИИ для создания архитектурных документов и проектных решений, такой документ гладко читается, содержит все положенные разделы и ни одного решения. Решение – это всегда отказ от альтернативы и принятая цена, а в сгенерированном тексте цены нет. Признак виден сразу: нет отброшенных вариантов и нет ответа, что сломается первым при росте нагрузки в десять раз. Лечится не запретом, а устной защитой: автор объясняет выбор, иначе документ не принимается. Оформление инструмент ускоряет, мышление не заменяет.

4. С посылкой не соглашусь: управление промптами не профессия, ему учат за неделю. Подорожал обратный навык: отличать правдоподобное от правильного, а он стоит на базе, поэтому требования мы не снижали: SQL и план запроса, сеть, конкурентность, структуры данных. Без базы разработчик с ИИ не быстрее, а опаснее. В онбординге центр тяжести переехал с «напиши» на «разберись»: задачи, на которых джуниоры росли неделями, закрываются за минуту, а с ними исчезает опыт отладки. Поэтому даём больше диагностики и требуем объяснить каждую строку и «так предложила модель» ответом не считается.

5. Главной проблемой для ИТ-департаментов, может стать исчезновение носителя знания: код есть, а человека, который помнит, почему система устроена именно так, нет. Пока всё работает, это незаметно, замечают в инциденте. Это первое.

Второе: когда писать дёшево, кода становится больше, чем нужно задаче, а стоимость владения не падает – бюджет поддержки растёт быстрее бюджета разработки. Плюс лицензии и небезопасные зависимости, за это отвечает компания, а не инструмент. БИТ

«Главный риск я вижу в размывании ответственности: код, который ты не писал, а только просмотрел, психологически перестаёт быть твоим. Чувство владения кодом естественным образом слабеет. А вместе с ним слабеет и внимание к тому, что уходит в прод. Страховаться нужно уже сейчас, не дожидаясь, пока это станет проблемой»

Эдуард Тихомиров,
руководитель разработки GitFlic (входит в "Группу Астра")

В работе разработчиков с распространением ИИ заметно изменился инструментальный слой. GitHub, GitLab, GitFlic из хранилищ кода превращаются в полноценные среды работы с ним – с ассистентами, автоматическим ревью, генерацией и проверками прямо в пайплайне. А вот суть работы осталась прежней. Разработчик всё так же разбирается в требованиях, принимает архитектурные решения, выбирает компромиссы и отвечает за то, что уходит в прод.

Изменилось соотношение: меньше механического набора кода, больше постановки задачи и проверки результата. При этом разрыв в производительности между теми, кто уже встроил ИИ в свой рабочий процесс, и теми, кто ещё нет, уже вполне заметен – и он будет только увеличиваться.

Ситуация, когда разработчики генерируют с помощью ИИ не только код, но и тесты к нему – у нас это уже норма. ИИ неплохо «отыгрывает» пользователя: воссоздаёт пользовательский опыт, придумывает нелогичные, но вполне реальные сценарии, до которых сам разработчик не додумается, потому что знает, «как надо».

С ростом доступности инструментов все чаще разработчики используют ИИ для создания архитектурных документов и проектных решений

Для black-box тестирования заходит идеально. С юнит-тестами чуть сложнее, но тоже лишними не будут. Ключевое – промпт. Если просто сказать «напиши тесты», получишь ассерты на happy path и ощущение покрытия вместо покрытия. А если развернуть контекст и попросить именно «придумай, как это сломать» – на выходе вполне вменяемые кейсы: граничные значения, гонки, некорректные входные данные.

С ростом доступности инструментов все чаще разработчики используют ИИ для создания архитектурных документов и проектных решений. Документ, который раньше требовал дня работы, теперь генерируется за час. Внешне всё складно: структура правильная, терминология на месте. Проблема вскрывается на этапе реализации, когда выясняется, что решение не учитывало конкретные ограничения системы или просто воспроизводило общий случай без адаптации. Мы ввели практику коротких технических ревью архитектурных предложений, чтобы отделить «красиво написано» от «реально работает».

Базовые требования к junior-разработчикам не снизились, изменился акцент. Раньше junior должен был уметь написать код. Сейчас важнее, чтобы он понимал, почему код написан именно так, потому что проверять ИИ-генерацию без этого понимания невозможно. Онбординг стал включать больше разборов чужого кода: junior учится не генерировать, а задавать правильные вопросы к результату. Умение промптить полезно, но без инженерной основы оно не работает.

Что может стать главной проблемой для ИТ-департаментов, когда ИИ перейдёт из статуса «помощника» в статус основного автора кода? На самом деле значительная часть кода уже пишется ИИ, так что переход скорее количественный, чем качественный.

Главный риск я вижу в размывании ответственности: код, который ты не писал, а только просмотрел, психологически перестаёт быть твоим. Это не лень – просто чувство владения кодом естественным образом слабеет, когда ты не автор, а ревьюер. А вместе с ним слабеет и внимание к тому, что уходит в прод. Отсюда вывод: страховаться нужно уже сейчас, не дожидаясь, пока это станет проблемой.

Во-первых, наращивать автоматический контроль – статические анализаторы, линтеры, проверки безопасности, мутационное тестирование: всё, что ловит проблемы без участия уставшего человека.

Во-вторых, сохранять обязательное независимое ревью – в том числе с привлечением тех же нейросетей как второго мнения, но при этом ответственность за мердж всегда остаётся на конкретном человеке, а не на инструменте. БИТ

«Если человек не может конкретизировать задачу, то и документ ТЗ получается с расплывчатыми формулировками. Но если человек постарался и дал хорошую базу для работы ИИ, то на выходе получится на 90% готовый полноценный документ»

Леонид Дегтярев,
директор по информационным технологиям ГК X-Com

1. Лично я расцениваю ИИ скорее как помощника, которого можно привлечь, например, чтобы быстрее найти слабые места, посмотреть на какую-то проблему с новой стороны «незашоренным» взглядом. При этом я задействую разные инструменты ИИ, и они анализируют множество источников. Искусственный интеллект позволяет с бо́льшим кругозором подходить и анализировать «узкие места» и области, которые трудно быстро понять самостоятельно. В целом, ИИ-инструменты помогают как минимум найти и накидать варианты решения вашей задачи, провести своеобразный «ускоренный брейншторм».

2. Да, мы сталкиваемся в своей команде с ситуацией, когда разработчики генерируют с помощью ИИ не только код, но и тесты к нему. И относимся к этому спокойно. Это нормальная практика. А почему нет? С другой стороны, ответственный исполнитель все равно должен сам описать сценарии для тестов, чтобы затем, неважно с помощью ИИ или нет, можно было правильно попробовать инструмент, который создает разработчик.

Главное наше опасение, как команд разработки и поддержки – не упустить тот момент, когда человек перестанет владеть информацией, которая была заложена в коде

3. Честно говоря, для самой разработки ИИ пока не очень подходит, пользуются нечасто. Гораздо чаще мы сталкиваемся с тем, что продуктовые либо бизнес-команды пытаются через ИИ сгенерировать ТЗ, либо некие функциональные требования – это повсеместное явление. Не могу сказать, хорошо это или плохо.

На мой взгляд, все сильно зависит от того, кто формирует промт для ИИ. Если человек сам не может конкретизировать задачу, то и документ ТЗ получается с расплывчатыми формулировками и, по сути, бесполезный: много времени уходит на то, чтобы его расшифровать. Но если человек постарался и дал хорошую базу для работы ИИ, то на выходе получится на 90% готовый полноценный документ.

4. Скажу так: есть некая база, которую любой из сотрудник наших ИТ-подразделений должен знать и уметь. И без этой базы он не сможет составить и промт для ИИ.

5. Главное наше опасение, как команд разработки и поддержки – не упустить тот момент, когда человек перестанет владеть информацией, которая была заложена в коде. Это важный момент – чтобы человек, когда он пишет и выстраивает алгоритм, знал и понимал, как он архитектурно работает.

Да, когда ты сразу забираешь готовый результат, с одной стороны, это дает ускорение каких-то процессов, относительно неплохое качество и соблюдение стандартов разработки. Но при этом может пострадать согласованность и функциональность самого механизма в целом. Иногда всё-таки нужны когнитивные способности кого-то, кто посмотрит сверху и, возможно, сделает какие-то допущения. Но это будет осознанно. БИТ

Ключевые слова: ИИ, прод, код, автор кода, нейросеть, разработка, промпт, онбординг.