УДК 004.832 https://doi.org/10.33619/2414-2948/119/10

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ В ПРОЦЕССЕ РАЗРАБОТКИ: ПРАКТИЧЕСКИЕ СЦЕНАРИИ

©Омаралиев А. Ч., *ORCID: 0009-0000-9214-7488, SPIN-код: 8744-1113, канд. пед. наук, Ошский государственный университет, г. Ош, Кыргызстан, aomaraliev@oshsu.kg*
©Омаралиева Г. А., *ORCID: 0000-0003-1862-2142, SPIN-код: 4741-5012, канд. физ.-мат. наук, Ошский государственный университет, г. Ош, Кыргызстан, gulya@oshsu.kg*
©У Минг Юй, *Ошский государственный университет, г. Ош, Кыргызстан,1969924852@qq.com*
©Акимова Ч. А., *ORCID: 0009-0007-7880-5377, Ошский государственный университет, г. Ош, Кыргызстан, colponajakimova902@gmail.com*

ARTIFICIAL INTELLIGENCE IN DEVELOPMENT: PRACTICAL SCENARIOS

©Omaraliev A., *ORCID: 0009-0000-9214-7488, SPIN-code: 8744-1113, Ph.D., Osh State University, Osh, Kyrgyzstan, aomaraliev@oshsu.kg*
©Omaralieva G., *ORCID: 0000-0003-1862-2142, SPIN-code: 4741-5012, Ph.D., Osh State University, Osh, Kyrgyzstan, gulya@oshsu.kg*
©Wu Ming Yu, *Osh State University, Osh, Kyrgyzstan, 1969924852@qq.com*
©Akimova Ch., *ORCID: 0009-0007-7880-5377, Osh State University, Osh, Kyrgyzstan, colponajakimova902@gmail.com*

Аннотация. Описаны прикладные способы использования ИИ‑ассистентов в инженерной практике: формализация требований, генерация заготовок кода, поддержка SQL/миграций, локализация дефектов по логам, авто генерация тестов, актуализация документации и оптимизация CI/CD. Показаны типичные эффекты (экономия времени, снижение дефектов), риски (недостоверность, утечки, лицензии) и предложена простая методика оценки «до/после» на реальных задачах команды. Даны рекомендации для поэтапного внедрения.

Abstract. The paper presents practical applications of AI assistants in everyday software engineering: requirement formalization, code scaffolding, SQL/migrations support, log‑based fault localization, test generation, documentation upkeep, and CI/CD optimization. We summarize observable benefits (time savings, defect reduction), typical risks (hallucinations, data leakage, licensing), and a lightweight evaluation method based on before/after measurements on real backlog items. Implementation guidelines for phased adoption are provided.

Ключевые слова: ИИ-ассистенты, искусственный интеллект, LLM; программная инженерия, тестирование, CI/CD, документация как код.

Keywords: AI assistants, artificial intelligence, LLM, software engineering, testing; CI/CD, docs‑as‑code.

Т. 11. №10 2025
https://doi.org/10.33619/2414-2948/119

ИИ-ассистенты уже стали повседневным инструментом разработчиков: помогают быстрее «снимать разгон» по задаче, держать в порядке тесты и документацию, разбирать логи и подсказывать мелкие правки в пайплайнах. При этом реальный эффект далёк от рекламных обещаний: модель не пишет продукт вместо команды и не принимает архитектурные решения. Она экономит усилия там, где работа стандартизируема и результат поддаётся проверке. Цель этого текста – дать практичную, воспроизводимую картину применения ассистентов и показать, как честно измерять пользу без академического переусложнения. Мы обращаемся к двум аудиториям: инженерам, которым нужна опора в ежедневной рутине, и руководителям, которые отвечают за качество и скорость релизов. В центре внимания — конкретные точки процесса, где ИИ действительно «подхватывает»: формализация разрозненных требований в удобный для согласования черновик, генерация заготовок кода и тестов, подготовка SQL-миграций, поддержка пайплайнов CI/CD и аккуратная синхронизация документации с изменениями в коде. Речь не о «замене человека», а о сохранении внимания на том, что требует опыта: разговор с заказчиком, выбор архитектуры, вдумчивое ревью сложной логики. Под «ИИ-ассистентом» мы понимаем три привычных формы: встроенные в IDE помощники, диалоговые инструменты (чат) и интеграции через API внутри командных ботов и CI. Мы сознательно не уходим в автономных «агентов» и академические бенчмарки — фокус на производственной практике. Отдельно проговариваем границы: защита данных, лицензии и контроль качества остаются ответственностью команды; любые генерации проходят через обычные инженерные проверки — тесты, статанализ, код-ревью. Метод оценки простой и повторяемый: короткие эксперименты «до/после» на реальных задачах бэклога. Мы фиксируем артефакты, промпты, итоговый код и длительность выполнения; смотрим на три оси — время, качество, стоимость. Важно отделять удобство восприятия от объективных показателей и масштабировать только те практики, которые стабильно дают прирост. Такой подход позволяет говорить с бизнесом на понятном языке и принимать решения не «по ощущениям», а на основе данных.

Вклад работы — в оперативном наборе рекомендаций: карта сценариев с ожидаемыми эффектами, минимальные примеры и кейсы, риски и способы их смягчения, а также лёгкая методика измерений, которую можно внедрить без перестройки процесса. Код приводится лишь там, где он иллюстрирует мысль, — акцент остаётся на организации труда и инженерной культуре. Рисунки и таблицы служат ориентиром для команды, помогая быстро найти, с чего начать и как убедиться, что ассистент действительно экономит время и делает продукт надёжнее.

Современная повестка по применению ИИ в разработке складывается из двух потоков: (а) фундаментальные работы по ИИ и методам построения интеллектуальных систем, задающие понятийный аппарат и рамки корректности; (б) прикладные исследования и отраслевые обзоры по использованию больших языковых моделей в реальных инженерных процессах — генерации кода и тестов, поддержке документации, улучшении пайплайнов CI/CD и качественных практик команды. Классический труд Р. Рассела и П. Норвига фиксирует эволюцию подходов к представлению знаний, поиску и обучению, напоминая, что успешное применение ИИ в инженерии опирается на формализацию целей, ограничений и критериев качества, а не на «магические» эвристики [1]. Это важно и для LLM-ассистентов: они дают выигрыш ровно там, где результат формализуем и проверяем – через автотесты, статический анализ и четкие контракты API. Русскоязычный массив публикаций последних лет систематизирует практические сценарии использования ИИ в программировании: ускорение написания типового кода, автоматизация тест-дизайна, улучшение качества через

Т. 11. №10 2025
https://doi.org/10.33619/2414-2948/119

подсказки ревью и статанализ, а также риски — от недостоверных ответов до лицензионной и конфиденциальной проблематики. В обзорных и прикладных статьях (Символ науки, Научный альманах, Universum: Технические науки, Молодой учёный) фиксируются ожидаемые эффекты (сокращение времени, рост покрытия тестами) и указываются условия воспроизводимости (наличие спецификаций, схем данных, формализованных критериев приемки) [2-4].

Параллельно англоязычные и смешанные источники (открытые журналы и практико-ориентированные публикации) поднимают вопрос методик измерения: сравнение «до/после» на реальных задачах и калибровка метрик под конкретный контур разработки. Промышленная практика не ограничивается генерацией кода. Отраслевые материалы подчёркивают пользу ассистентов в поддержании «инженерной гигиены»: синхронизация документов с изменениями в коде, реструктуризация пайплайнов, кэширование и сборка, разметка логов для быстрой локализации дефектов. Эти аспекты регулярно поднимаются в профессиональных публикациях и аналитике (включая русскоязычные медиа и отраслевые обзоры), где акцент делается на измеримые выгоды — снижение P95 времени сборки, уменьшение числа возвратов на ревью, стабилизация скорости релизов [5].

Отдельная линия — архитектурный контекст. Решения уровня «монолит микросервисы определяют точки встраивания ассистентов и границы их пользы: где уместны генерация каркасов и контрактов, где автоматизация тестов, где поддержка эксплуатационных практик. Соответствующая литература по архитектуре микросервисов (в русском издании) предлагает критерии выбора, которые хорошо соотносятся с «врезками» ИИ в процесс: там, где есть чёткие интерфейсы и стандартизируемые шаги, ассистент ускоряет работу и снижает вариативность результата. В локальном и смежном контексте полезны работы, не напрямую посвящённые LLM, но задающие инженерную планку и демонстрирующие важность производительности и корректности вычислений. Например, исследование многопоточной обработки данных в C#/SQLite показывает, что выгоды инструментов (в т. ч. ассистентов) раскрываются только вместе с дисциплиной производительности, тестирования и контроля состояния данных [6]. Публикации по прикладной математике и численным иллюстрируют зрелость научной среды, в которой востребованы формальные постановки и проверяемые решения – это важный фон для ответственного внедрения ИИ-инструментов.

Основные практические сценарии. Формализация требований из свободного текста. Ассистент по переписке/брифу предлагает use-cases, ограничения, примеры и негативные сценарии. Выход — черновик спецификации для согласования.

Архитектурные эскизы под нефункциональные требования. Краткое сравнение «монолит – микросервисы – serverless» с рисками, точками роста и влиянием на SLO/SLA.

Заготовки кода и типовые каркасы. По описанию интерфейса генерируются контроллеры, DTO, валидация, базовые тесты. Сохранение стиля проекта обязательно.

SQL/ORM-запросы и миграции. Ассистент синтезирует запросы по схеме БД и ТЗ, объясняет индексы; инженер проверяет планы выполнения и транзакционность.

Локализация дефектов по логам и трассировкам. По стектрейсу/логам формируются гипотезы причин и минимальный сценарий воспроизведения.

Генерация юнит-/интеграционных тестов и тест-дизайн E2E. Рост покрытия, лучшее покрытие «краёв», чек-листы в Gherkin.

Актуализация документации из diff. Предлагаются правки в README, API-справке и changelog на основе изменений в коде.

Оптимизация DevOps/CI/CD. Подсказки для Dockerfile, кэширование зависимостей, разбиение стадий, уменьшение базовых образов.

Т. 11. №10 2025
https://doi.org/10.33619/2414-2948/119

Таблица

КАРТА СЦЕНАРИЕВ ПРИМЕНЕНИЯ ИИ В ПРОЦЕССЕ РАЗРАБОТКИ

Постановка задачи	*Реализация*	*Тестирование*	*Сопровождение*
Формализация требований	Заготовки кода	Генерация тестов	Документация из diff
Архитектурные эскизы	SQL/миграции	Тест-дизайн e2e	Помощник в код-ревью
	Рефакторинг легаси	Тестовые данные	CI/CD оптимизация
			Локализация дефектов

Методика оценки эффективности. Материал: 10–20 задач из бэклога, сопоставимых по сложности. Два режима исполнения: «без ИИ» и «с ИИ». Фиксируются артефакты, подсказки, итоговый код, прохождение тестов, длительность. Ключевые метрики: время на задачу; доля задач, прошедших тесты с первой попытки; замечания на код-ревью; дефекты; прямые затраты на ИИ. Эффект: относительное сокращение времени, цена «правильной задачи», стабильность (снижение разброса).

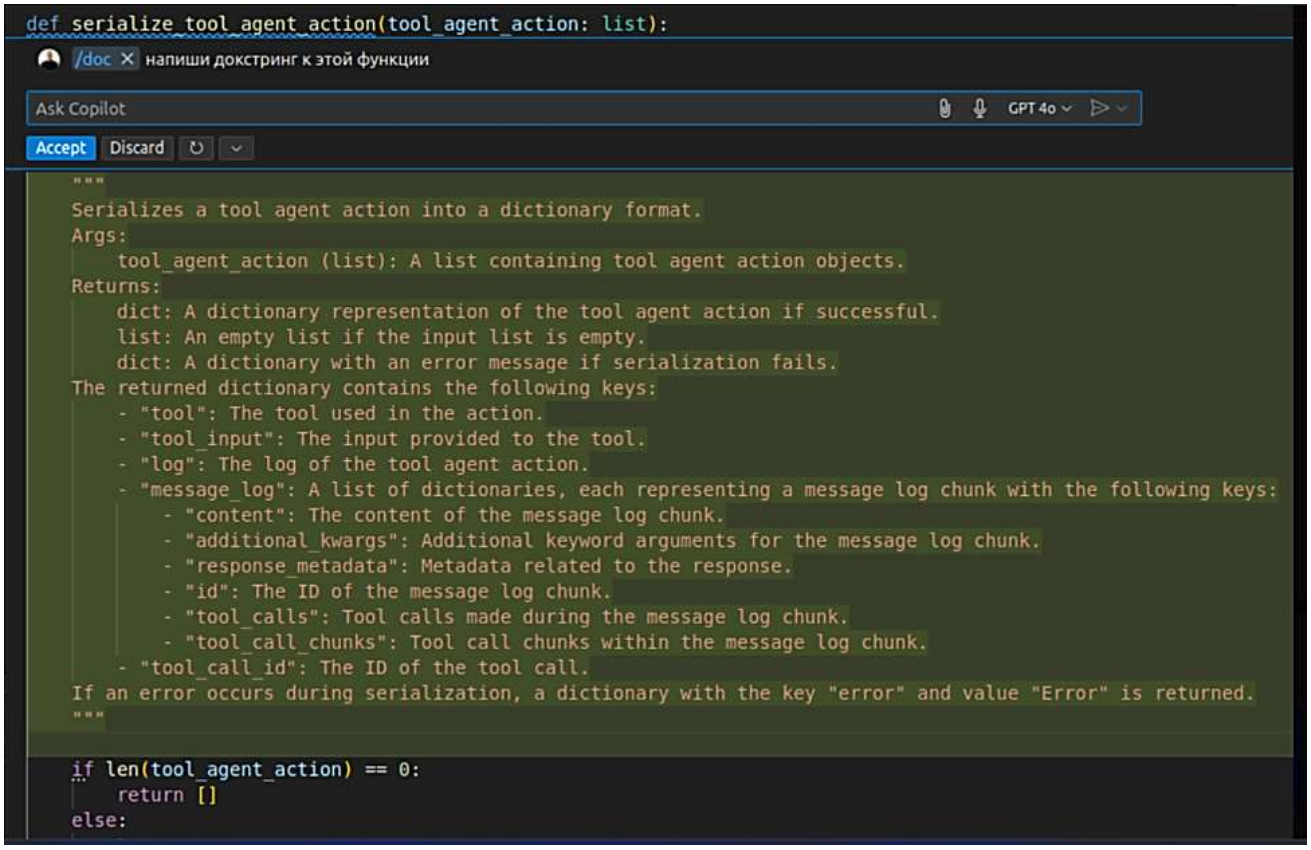

Рисунок 1. Интеграция ИИ-ассистента в IDE

Мини-Кейсы. Генерация тестов в легаси-модуле расчётов. Ассистент предложил набор кейсов, команда отобрала релевантные и добавила фикстуры: выросло покрытие, сократились замечания на ревью. Ускорение пайплайна CI. Рекомендации по кэшированию и разделению стадий снизили p95 времени сборки; стало проще выпускать мелкие патчи.

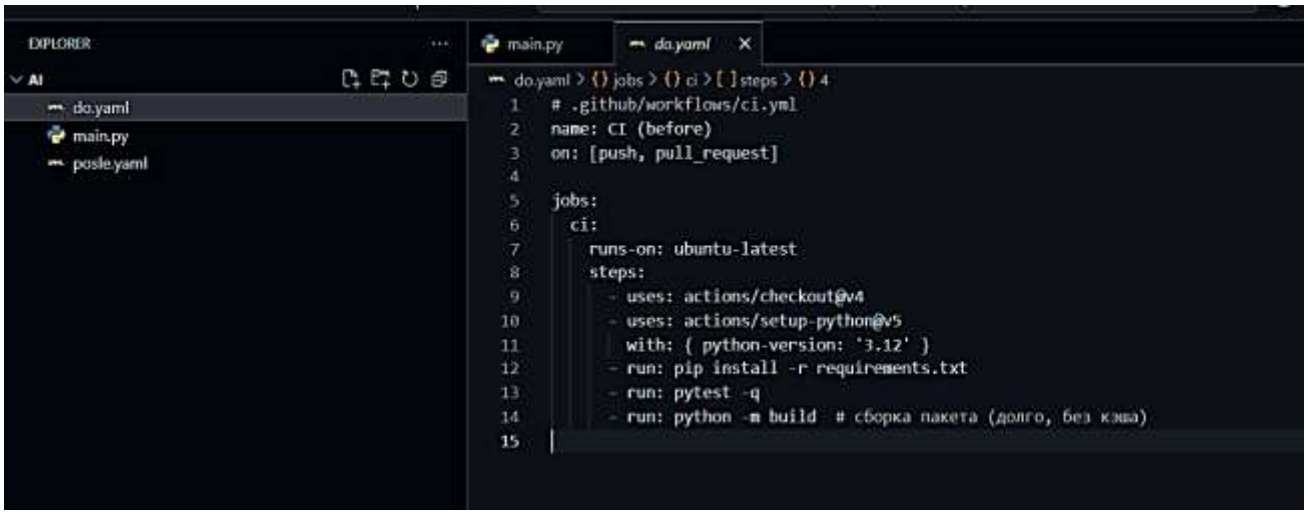

Рисунок 2. Улучшение пайплайна CI: до (кэширование, артефакты, параллельные этапы)

Риски и ограничения. Недостоверные ответы — лечатся тестами и правилом «модель пишет — инженер проверяет». Опасность утечки — ограничиваем данные и анонимизируем. Лицензии – проверяем совместимость и ведём журнал генераций. Зависимость от провайдера — держим план B. Не перегибаем с автоматизацией, сохраняем творческую часть работы инженера.

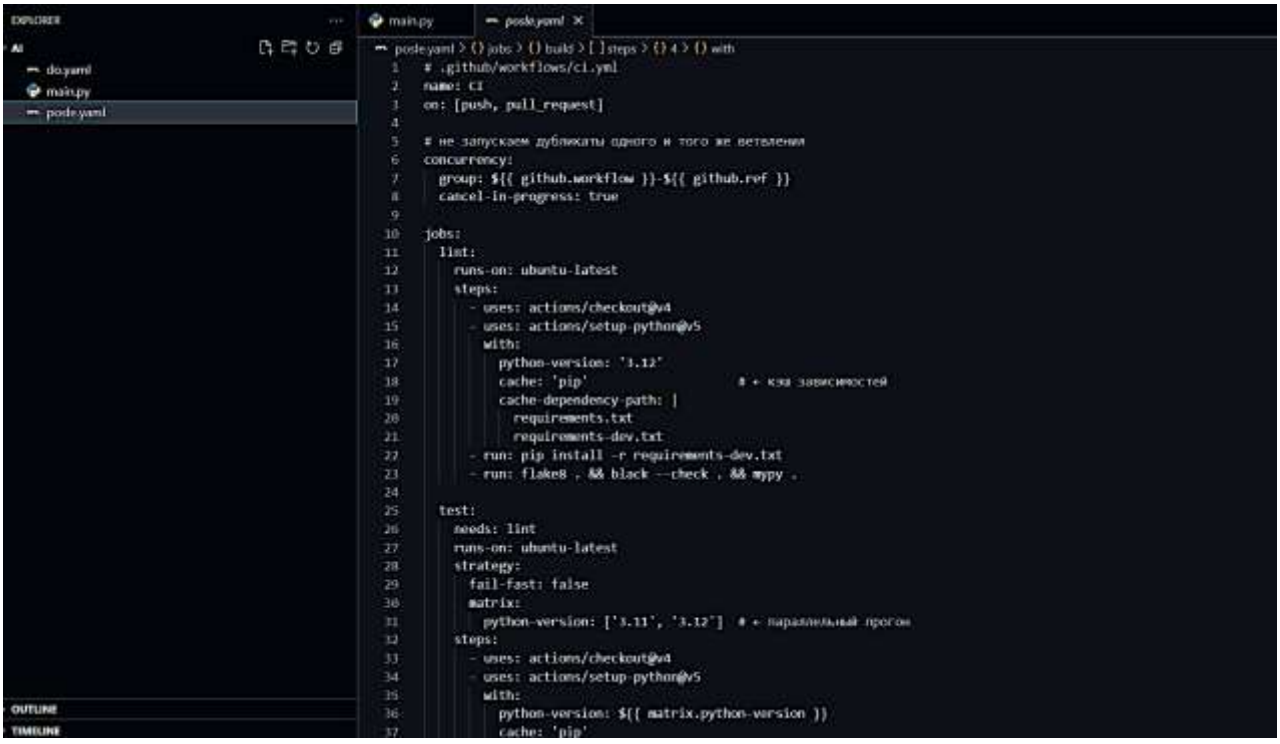

Рисунок 3. Улучшение пайплайна CI: после

S3. Каркас rest-ресурса на fastapi (python)
# app/main.py
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel, Field, condecimal
from typing import List

Т. 11. №10 2025
https://doi.org/10.33619/2414-2948/119

```
app = FastAPI(title="Billing API")
class InvoiceItem(BaseModel):
    sku: str = Field(min_length=1)
    qty: int = Field(gt=0)
    price: condecimal(gt=0)
class Invoice(BaseModel):
    id: str
    customer_id: str
    items: List[InvoiceItem]
FAKE_DB = {}
@app.post("/invoices", status_code=201)
async def create_invoice(inv: Invoice):
    if inv.id in FAKE_DB:
        raise HTTPException(status_code=409, detail="Invoice exists")
    # TODO: бизнес-логика, валидация скидок и налогов
    FAKE_DB[inv.id] = inv.model_dump()
    return {"ok": True, "id": inv.id}
@app.get("/invoices/{inv_id}")
async def get_invoice(inv_id: str):
    if inv_id not in FAKE_DB:
        raise HTTPException(status_code=404, detail="Not found")
    return FAKE_DB[inv_id]
# tests/test_invoices.py
from fastapi.testclient import TestClient
from app.main import app
client = TestClient(app)
def test_create_and_get_invoice():
    payload = {
        "id": "INV-1",
        "customer_id": "CUST-9",
        "items": [{"sku": "SKU1", "qty": 2, "price": "10.50"}]
    }
    r = client.post("/invoices", json=payload)
    assert r.status_code == 201
    r = client.get("/invoices/INV-1")
    assert r.status_code == 200
    body = r.json()
    assert body["customer_id"] == "CUST-9"
```

S4. SQL И МИГРАЦИИ (POSTGRESQL + ALEMBIC)

```
-- db/schema.sql
CREATE TABLE customers (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    email CITEXT UNIQUE,
    created_at TIMESTAMPTZ DEFAULT now()
);
CREATE TABLE invoices (
```

Т. 11. №10 2025
https://doi.org/10.33619/2414-2948/119

```
    id TEXT PRIMARY KEY,
    customer_id UUID NOT NULL REFERENCES customers(id),
    total NUMERIC(12,2) NOT NULL CHECK (total >= 0),
    created_at TIMESTAMPTZ DEFAULT now()
);
CREATE INDEX idx_invoices_customer_created
    ON invoices(customer_id, created_at DESC);
-- пример запроса
EXPLAIN ANALYZE
SELECT * FROM invoices
WHERE customer_id = '2b8c…-uuid' AND created_at >= now() - interval '30 days'
ORDER BY created_at DESC
LIMIT 50;
# migrations/2025_09_03_add_invoice_items.py (Alembic)
from alembic import op
import sqlalchemy as sa
revision = "2025_09_03_add_invoice_items"
down_revision = None
def upgrade():
    op.create_table(
        "invoice_items",
        sa.Column("invoice_id", sa.Text, sa.ForeignKey("invoices.id"), primary_key=True),
        sa.Column("line_no", sa.Integer, primary_key=True),
        sa.Column("sku", sa.Text, nullable=False),
        sa.Column("qty", sa.Integer, nullable=False),
        sa.Column("price", sa.Numeric(12, 2), nullable=False),
    )
    op.create_index("idx_items_invoice", "invoice_items", ["invoice_id"])
def downgrade():
    op.drop_index("idx_items_invoice", table_name="invoice_items")
    op.drop_table("invoice_items")
```

ИИ-ассистенты не заменяют инженера, но заметно ускоряют работу там, где есть повторяемые операции и чёткие критерии качества. Практика показывает: выигрыши приходят от «мелочей» – каркасы модулей, покрытие «краёв» тестами, аккуратные миграции, порядок в пайплайнах. Освободив руки от рутины, команда возвращает внимание к тому, что требует опыта и ответственности: разговору с заказчиком, архитектурным решениям, вдумчивому ревью сложной логики. Устойчивый эффект возникает не за счёт «мощности» модели, а благодаря дисциплине процесса. Там, где есть автотесты, статический анализ, прозрачные контракты API и живая документация, ИИ работает как мультипликатор продуктивности; где этого нет – лишь ускоряет накопление технического долга. Поэтому внедрение должно быть поэтапным и измеримым: начинать с узких, хорошо проверяемых сценариев, фиксировать показатели «до/после» (время, доля прохождения тестов с первой попытки, замечания на ревью, прямые затраты), оставлять в потоке только те практики, которые стабильно дают прирост.

Риски управляемы там, где действует «контур безопасности»: минимизация и анонимизация данных, проверка лицензий генерируемого кода, журналирование

взаимодействий с ассистентом, обязательная верификация результата человеком. В доменах с высокой ценой ошибки нужны повышенные требования к проверкам и правило «двух пар глаз». В таком подходе ИИ занимает естественное место рядом с системой контроля версий и CI: помогает беречь внимание инженера и делать продукт надёжнее. Следующий практический шаг — закрепить успешные сценарии в регламентах команды, поддерживать каталог удачных примеров и промптов, периодически пересматривать метрики. Это позволяет масштабировать пользу без потери качества и строить зрелую инженерную культуру, в которой ИИ – инструмент, а не самоцель.

Список литературы:

1. Рассел С., Норвиг П. Искусственный интеллект: современный подход. М.: Вильямс, 2021. 1152 с.

2. Бевзенко С. А. Исследование методов автоматического программирования с применением искусственного интеллекта // Молодой ученый. 2024. №11(510). С. 13-15.

3. Гылыджова А. Б., Оразгельдыева А. Обзор кода на основе искусственного интеллекта: новый подход к улучшению качества программного обеспечения // Символ науки. 2024. №12-1-2. С. 106-107.

4. Грызлов Д. В., Куваева Е. Н. Использование искусственного интеллекта в программировании // Научный аспект. 2024. №8. С. 2121-2127.

5. Федорова П. В., Данилин В. А., Мардамшина А. А. Развитие и применение искусственного интеллекта в программировании // Вестник науки. 2024. Т. 4. №10 (79). С. 806-810.

6. Маличенко С. В. Проблемы перехода от монолитной к микросервисной архитектуре // Евразийский научный журнал. 2022. №5. С. 8-19.

7. Казырский Н. А. Как ИИ меняет программирование: угрозы и возможности // Научный лидер. 2025. №11(212).

8. Токторбаев А. М., Карабаев С. Э., Мойдунова А. С., Абдыкадыров С. К. Многопоточная обработка данных в SQLite с использованием C# // Engineering problems and innovations. 2025. Т. 3. №2. С. 16-23.

9. Ньюмен С. Создание микросервисов. СПб.: Питер, 2023.

10. Петровский А. С. Применение искусственного интеллекта в приборостроении // Universum: технические науки. 2024. Т. 2. №1 (118). С. 10-14. https://doi.org/10.32743/UniTech.2024.118.1.16682

11. Gartner: к 2027 году генеративный ИИ потребует повышения квалификации 80% инженеров // Открытые системы. СУБД. 2024.

References:

1. Rassel, S., & Norvig, P. (2021). Iskusstvennyi intellekt: sovremennyi podkhod. Moscow. (in Russian).

2. Bevzenko, S. A. (2024). Issledovanie metodov avtomaticheskogo programmirovaniya s primeneniem iskusstvennogo intellekta. *Molodoi uchenyi*, (11(510)), 13-15. (in Russian).

3. Gylydzhova, A. B., & Orazgel'dyeva, A. (2024). Obzor koda na osnove iskusstvennogo intellekta: novyi podkhod k uluchsheniyu kachestva programmnogo obespecheniya. *Simvol nauki*, (12-1-2), 106-107. (in Russian).

4. Primenenie iskusstvennogo intellekta v programmirovanii (2024). *Nauchnyi al'manakh*, (8), 8–10. (in Russian).

5. Fedorova, P. V., Danilin, V. A., & Mardamshina, A. A. (2024). Razvitie i primenenie iskusstvennogo intellekta v programmirovanii. *Vestnik nauki, 4*(10 (79)), 806-810. (in Russian).

6. Malichenko, S. V. (2022). Problemy perekhoda ot monolitnoi k mikroservisnoi arkhitekture. *Evraziiskii nauchnyi zhurnal,* (5), 8-19. (in Russian).

7. Kazyrskii, N. A. (2025). Kak II menyaet programmirovanie: ugrozy i vozmozhnosti. *Nauchnyi lider,* (11(212)). (in Russian).

8. Toktorbaev, A., Karabaev, S., Moidunova, A., & Abdykadyrov, S. (2025). Mnogopotochnaya obrabotka dannykh v SQLite s ispol'zovaniem C. *Engineering problems and innovations, 3*(2), 16-23. (in Russian).

9. N'yumen, S. (2023). Sozdanie mikroservisov. St. Petersburg. (in Russian).

10. Petrovskii, A. S. (2024). Primenenie iskusstvennogo intellekta v priborostroenii. *Universum: tekhnicheskie nauki, 2*(1(118)), 10-14. (in Russian). https://doi.org/10.32743/UniTech.2024.118.1.16682

11. Gartner: k 2027 godu generativnyi II potrebuet povysheniya kvalifikatsii 80% inzhenerov (2024). Otkrytye sistemy. SUBD. (in Russian).

Поступила в редакцию
13.09.2025 г.

Принята к публикации
21.09.2025 г.

Ссылка для цитирования:

Омаралиев А. Ч., Омаралиева Г. А., У Минг Юй, Акимова Ч. А. Искусственный интеллект в процессе разработки: практические сценарии // Бюллетень науки и практики. 2025. Т. 11. №10. С. 74-82. https://doi.org/10.33619/2414-2948/119/10

Cite as (APA):

Omaraliev, A., Omaralieva, G., Wu Ming, Yu, & Akimova, Ch. (2025). Artificial Intelligence in Development: Practical Scenarios. *Bulletin of Science and Practice, 11*(10), 74-82. (in Russian). https://doi.org/10.33619/2414-2948/119/10