5.2. Управление ожиданиями и «Low-Hanging Fruit»

Управление ожиданиями

- **Начало пути будет неровным.** Когда команда впервые пробует LLM-инструменты, производительность часто падает: нужно время, чтобы освоить новые рабочие процессы, подобрать подходящие модели и отладить пайплайны. Лишь после периода обучения и совершенствования процессов появляется ощутимый рост.
- **Постоянная шлифовка.** Модели и инструменты быстро меняются. Требуется время на настройку запросов, корректную интеграцию `retrieval`-слоя и построение систем контроля качества. Первоначальная цель — не мгновенный прирост, а постепенное улучшение стандартов кода.
- **Роль руководства.** На практике команды быстрее начинают работать с ИИ, если лидеры активно используют инструменты и демонстрируют выгоды. Отношение тимлида к внедрению напрямую влияет на готовность разработчиков экспериментировать с LLM.

LLM как «дешёвый аналитик легаси»

В контексте устаревших, плохо документированных систем самые явные выгоды ИИ-ассистентов связаны с анализом и описанием существующего кода.

Генерация документации.

Проекты вроде `Autodoc` автоматически обходят репозиторий, индексируют код и вызывают LLM для генерации документации по каждому файлу и папке[1]. Документация живёт рядом с кодом, путешествует вместе с репозиторием и доступна через простой CLI-интерфейс для ответов на вопросы разработчиков[1]. Коммерческие решения, такие как Cosine AutoDoc, создают обзорные страницы, глубокие описания модулей, API-справки и примеры кода без аннотаций и поддерживают их в актуальном состоянии с каждым коммитом[2]. Они также визуализируют взаимодействия между классами и функциями, помогая понять, как части системы связаны[2]. По данным Cosine, 37 % разработчиков называют документацию задачей №1, где им нужна помощь ИИ[2].

Ориентирование в легаси-коде.

`AutoDoc` помогает командам понимать сложный легаси-код без долгого «спелеинга» — разработчики могут задавать вопросы и получать ответы с ссылками на конкретные файлы[2]. Это особенно полезно при модернизации или переписывании старых систем.

Автоматическое покрытие тестами.

Сгенерированные тесты позволяют безопасно доработать устаревший код. GitHub Copilot и Visual Studio Code предлагают набор возможностей: помощь в настройке тестовых фреймворков, генерацию `unit-`, интеграционных и `end-to-end`-тестов, создание тестов для граничных случаев и исправление падающих тестов[3]. В документации Copilot подчёркивается, что можно быстро создавать тесты через чат-промпты («Generate tests for this code», «Write unit tests including edge cases»), а также использовать интеллектуальные действия в редакторе для генерации тестового файла[3].

«Low-Hanging Fruit» для старта

- **Документация** (README, JSDoc, API-описание). Генерация документации — самый очевидный и наименее болезненный шаг. Инструменты вроде Autodoc и Cosine AutoDoc автоматически создают обзорные страницы, детализированные разделы и API-справки, держат их актуальными и помогают понять, как части системы связаны[2]. Внутренние проекты могут воспользоваться CLI-инструментами для индексации и запросов[1].
- **Генерация тестов для легаси-кода.** GitHub Copilot способен автоматически генерировать целевые `unit-` , проверяя основную функциональность, обработку входных данных, исключения и побочные эффекты. В примерном промпт-файле от GitHub описана стратегия: проверять ожидаемое поведение, тестировать типичные входы и граничные значения, проверять обработку ошибок и тесты побочных эффектов, следовать AAA-паттерну и писать независимые тесты[4]. В документации подчеркивается, что Copilot упрощает настройку тестовых фреймворков, генерацию тестов и исправление падающих тестов[3]; советы включают выделение кода для тестирования, точные и контекстные инструкции, тщательную проверку предложений и итеративный подход[5]. По сути, ИИ может взять на себя повторяющуюся работу, позволяя разработчикам сосредоточиться на сложной логике.
- **Переписывание и рефакторинг.** Хотя автоматическое переписывание целых модулей требует осторожности, LLM-ассистенты могут помочь предложениями по улучшению стиля и стандартизации. Использование протокола Model Context Protocol (MCP) позволяет подключать LLM к репозиториям и средам разработки, давая модели актуальный контекст для рефакторинга. В связке с автодокументированием и тестами это позволяет модернизировать легаси-код, избегая серьёзных регрессий.
- **Синтетические данные.** Новое направление — использовать LLM как фабрику синтетических данных для тестовых сценариев и `fuzz`-тестирования. Генеративные модели способны быстро создавать разнообразные входные данные, покрывая необычные случаи и сокращая трудозатраты на ручное создание наборов тестов.

Важные советы и лучшие практики

- **Тесты: фокус на ценность.** В примере промпта GitHub для генерации тестов предлагается генерировать 5–8 фокусных тестов, включать реалистичные данные, писать комментарии для сложной настройки, обеспечивать независимость тестов и концентрироваться на поведении функции, а не на деталях реализации[4]. Эти рекомендации помогают получить полезные тесты, а не просто «заполнить чекбокс».
- **Лучшие практики Copilot.** Авторы GitHub отмечают, что для успешной генерации тестов нужно выделять код, задавать конкретные вопросы, обеспечивать контекст

(комментарии и `docstring`), внимательно проверять предложения и итеративно уточнять запросы[5]. Также полезно спрашивать Copilot, какие тесты пропущены, и сочетать ИИ-инструмент с классическими инструментами покрытия, чтобы обнаруживать незатронутые участки кода[5].

- **Персонализация и консистентность.** Visual Studio Code позволяет задавать предпочтительные тестовые фреймворки, настраивать стиль тестов и уточнять генерацию с помощью команд `/setupTests` и `/generateTests`. Воспользуйтесь возможностью указывать, какие типы тестов нужны (`unit`, `integration`, `end-to-end`), какие `edge cases` учесть и какое именование использовать[3].

Итог

Управление ожиданиями и поиск «легких» побед важны для успешного внедрения LLM в инженерную практику. Начать стоит с задач, где ИИ уже даёт измеримую отдачу: автоматическое документирование кода, генерация тестов и модернизация легаси-систем. Эти направления помогают снять накопившийся долг, а также повысить качество и скорость разработки. Важную роль играет лидерство: когда тимлид сам использует и поддерживает ИИ-инструменты, команда охотнее экспериментирует и быстрее проходит стадию обучения. Параллельно стоит изучать новые подходы, такие как генерация синтетических данных, и готовиться к более сложным интеграциям LLM через MCP и агентные фреймворки.

Источники

[1] Источник: raw.githubusercontent.com
[2] Источник: cosine.sh
[3] Источник: code.visualstudio.com
[4] Источник: docs.github.com
[5] Источник: github.blog