# Программирование: как общаться с ИИ-агентом, чтобы код не пришлось переписывать

18.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

В 2026 году ИИ-агенты стали обычным инструментом разработчика: они пишут код, тесты, миграции и документацию. Но чем больше кода генерируется, тем дороже ошибки в постановке задачи. В этой статье — практический подход к формулировке задач для агента, который снижает количество переписываний и делает результат проверяемым.

## Почему «напиши фичу» не работает

ИИ-агент — это программа, которая принимает цель, контекст и ограничения, а затем предлагает изменения в коде. Без явных критериев готовности он оптимизирует то, что понял, а не то, что нужно. Типичные последствия: лишние зависимости, неверная обработка ошибок, отсутствие тестов и скрытые побочные эффекты.

Решение — относиться к агенту как к новому сотруднику: дать доступ к правилам проекта, показать примеры и договориться, как проверять результат.

## Что включить в постановку задачи

Хорошая задача для агента содержит пять блоков. Ниже — шаблон, который можно адаптировать под ваш процесс.

1. **Цель и контекст.** Опишите, какую проблему решаем и где именно в коде. Укажите файлы, модули и существующие паттерны.
2. **Ограничения.** Версии языка и фреймворков, запрещённые библиотеки, требования к стилю, лимиты по производительности.
3. **Критерии готовности.** Что должно работать, какие тесты добавить, какие сценарии проверить вручную.
4. **Формат результата.** Патч, список изменённых файлов, объяснение решений, план отката.
5. **Способ проверки.** Команды линтера, тестов и сборки, которые агент должен запустить сам.

   Правило: если критерий нельзя проверить командой или тестом, он не считается критерием готовности.

### Пример формулировки

Вместо «добавь кэширование» напишите: «Добавь кэш ответов API в модуле reports с TTL 60 секунд. Используй существующий клиент Redis из lib/cache. Добавь тест на попадание и промах кэша. Не меняй публичный интерфейс функции getReport. Запусти make test и make lint».

## Как передавать контекст

Агент не должен угадывать правила проекта. Передавайте их явно и в удобном для машины виде.

- **Файл правил.** Держите в репозитории документ с соглашениями: структура папок, обработка ошибок, логирование, работа с секретами.
- **Примеры кода.** Ссылайтесь на существующие модули, которые решают похожую задачу.
- **Схемы данных.** Приложите модели, контракты API и миграции, чтобы агент не придумывал поля.
- **История решений.** Кратко опишите, почему предыдущие подходы не подошли.

Чем меньше в задаче двусмысленностей, тем меньше циклов правок.

## Ревью результата: что проверять в первую очередь

Даже аккуратный агент может ошибаться. Сфокусируйте ревью на рисках, а не на стиле.

- **Границы изменений.** Агент не должен трогать файлы вне задачи без объяснения.
- **Зависимости.** Новые пакеты — только с обоснованием и проверкой лицензии.
- **Обработка ошибок.** Проверьте сетевые вызовы, таймауты и пустые ответы.
- **Безопасность.** Нет ли секретов в коде, логах и тестовых данных.
- **Тесты.** Покрывают ли они негативные сценарии, а не только счастливый путь.

Полезно просить агента отдельно перечислить допущения, которые он сделал. Это ускоряет поиск расхождений.

## Защита от галлюцинаций и устаревших знаний

Агент может ссылаться на несуществующие методы или устаревшие версии библиотек. Снизить риск помогают три приёма.

1. **Запрет на догадки.** Требуйте явно указывать, где не хватает информации, вместо выдумывания API.
2. **Проверка по исходникам.** Просите агента открыть файлы библиотеки в репозитории, а не полагаться на память.
3. **Малые шаги.** Дробите задачу на этапы: сначала интерфейс и тесты, затем реализация.

## Метрики, которые показывают пользу подхода

Чтобы понять, работает ли процесс, отслеживайте несколько простых показателей за спринт:

- доля задач, принятых без доработок;
- среднее число циклов правок на задачу;
- количество инцидентов, связанных с изменениями агента;
- время от постановки до мержа.

Если доля задач без доработок растёт, а инциденты не увеличиваются, формулировки стали точнее. Если наоборот — вернитесь к критериям готовности и контексту.

## Чек-лист перед отправкой задачи агенту

- Цель сформулирована одной фразой.
- Указаны файлы и модули, которых касается изменение.

- Перечислены ограничения по версиям и зависимостям.
- Есть проверяемые критерии готовности.
- Указаны команды тестов и линтера.
- Определён формат результата и план отката.

Такой подход не гарантирует идеальный код с первого раза, но делает работу с агентом предсказуемой. А предсказуемость в разработке ценнее скорости отдельного рывка.

Источник: https://cms.dolinais.ru/programmirovanie-kak-obschatsya-s-ii-agentom-chtoby-kod-ne-prishlos-perepisyvat