Программирование: как вести проект, когда код пишут ИИ-агенты

05.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

Почему это стало проблемой в 2026 году

В 2026 году ИИ-агенты вроде Claude Code и Codex пишут код целыми модулями, а иногда и проектами. Продакт или менеджер без технического бэкграунда может запустить разработку, не имея в команде программиста. Пример из практики: расширение для Chrome, 593 коммита и более 78 тысяч строк кода — всё создано двумя агентами. Но когда агент сообщает «Готово, все тесты проходят», проверить это некому. И тут начинается самое интересное: как убедиться, что код работает, не читая его построчно?

Эта статья — не про магию промптов. Она про **инженерную обвязку** вокруг агента: правила, автоматические проверки, приёмку и управление рисками. Материал будет полезен продактам, тимлидам и разработчикам, которые уже используют агентов или только планируют.

Что такое «обвязка» и зачем она нужна

Обвязка — это набор процессов, скриптов и договорённостей, которые заставляют агента работать предсказуемо и снижают вероятность ошибок. Без неё вы получаете чёрный ящик: агент генерирует код, вы не понимаете, что внутри, а тесты могут быть фиктивными.

Вот ключевые элементы обвязки, которые стоит внедрить с первого дня:

- **Правила (rules) —** жёсткие ограничения для агента: какие библиотеки можно использовать, какие нельзя, как оформлять коммиты, что делать запрещено (например, менять конфигурацию CI).
- **Проверки (checks) —** автоматические шаги после каждого изменения: линтеры, статический анализ, юнит-тесты, сборка. Агент должен запускать их сам и показывать результат.
- **Запреты (forbiddens) —** список действий, которые агент не может выполнять без явного разрешения: деплой в прод, изменение секретов, удаление файлов.
- **Скиллы (skills) —** заранее заготовленные сценарии для типовых задач: «добавить новую фичу», «исправить баг», «обновить зависимость». Это снижает вариативность и упрощает контроль.

Эти элементы описываются в конфигурационных файлах (например, .claude/rules.md или codex.rules) и подгружаются агентом перед началом работы. Чем детальнее правила, тем меньше сюрпризов.

Шесть шагов от идеи до приёмки

Процесс, который использовал продакт в упомянутом проекте, выглядит так:

1. **Формулировка задачи.** Чёткое описание того, что нужно сделать, с критериями приёмки. Без размытых формулировок вроде «улучшить UI».
2. **Декомпозиция.** Разбить задачу на мелкие шаги, каждый из которых можно проверить отдельно. Агент лучше справляется с узкими задачами.
3. **Генерация кода агентом.** Агент пишет код и запускает проверки. Если проверки не проходят, он

должен исправить ошибки самостоятельно, но с ограниченным числом попыток (например, не более трёх).
4. **Ревью артефактов.** Вы не читаете код, но проверяете артефакты: отчёт о тестах, логи сборки, diff изменений (можно смотреть только ключевые файлы), скриншоты работающего приложения.
5. **Приёмка.** Запуск готовой функциональности в тестовой среде. Если что-то не так, задача возвращается агенту с конкретным описанием проблемы.
6. **Фиксация результата.** Коммит с осмысленным сообщением, обновление документации (её тоже может генерировать агент, но проверять нужно вам).

Важно: на каждом шаге у вас должны быть **автоматические индикаторы**, которые нельзя подделать. Например, CI-пайплайн, который запускается на каждый коммит и показывает статус. Если агент говорит «тесты прошли», а CI молчит — это красный флаг.

Пять улучшений, которые стоит внести в следующий проект

Даже при хорошей обвязке остаются слабые места. Вот что рекомендует продакт, прошедший этот путь:

- **Внешний аудит кода.** Раз в спринт привлекать независимого разработчика для выборочного ревью. Это дорого, но дешевле, чем исправлять архитектурные ошибки потом.
- **Больше интеграционных тестов.** Юнит-тесты агент пишет хорошо, а вот взаимодействие модулей часто ломается. Нужны тесты, которые проверяют сценарии целиком.
- **Ограничение на размер изменений.** Один коммит — одна логическая правка. Так проще откатывать и анализировать.
- **Явные метрики качества.** Например, покрытие кода тестами (через CI), количество ошибок на тысячу строк, время сборки. Агент должен отчитываться по этим метрикам.
- **Регулярная ротация агентов.** Если один агент застрял, второй может предложить другое решение. Но не стоит смешивать их работу в одном коммите.

У каждого улучшения есть довод против: аудит стоит денег, интеграционные тесты замедляют разработку, ротация усложняет отслеживание. Но в долгосрочной перспективе эти меры снижают риск получить неуправляемый код.

Типичные ошибки и как их избежать

Вот что чаще всего идёт не так:

- **Доверие к отчётам агента без проверки.** Агент может «забыть» запустить тесты или сымитировать успех. Всегда смотрите на вывод CI.
- **Отсутствие правил.** Без ограничений агент может подключить незнакомую библиотеку или изменить конфигурацию сервера. Пропишите явные запреты.
- **Слишком большие задачи.** Чем больше контекст, тем выше вероятность ошибки. Дробите работу.
- **Игнорирование документации.** Агент генерирует код быстро, но документация отстаёт. Заставьте его обновлять README и комментарии.

  «Агент пишет: “Готово, все тесты проходят”. Дальше у меня два пути: поверить или показать тому, кто разбирается. Показать некому» — типичная ситуация в проектах без разработчика. Решение — автоматические проверки, которые нельзя обойти.

Инструменты и их роль

В 2026 году на рынке есть несколько агентов, но принципы работы с ними схожи. Claude Code и Codex — лишь примеры. Важно, чтобы агент поддерживал:

- загрузку правил из файлов;
- запуск внешних команд (тесты, линтеры);
- возможность ограничить доступ к файлам и сети;
- логирование действий для аудита.

Если агент не поддерживает что-то из этого, обвязку придётся строить вручную, например через скрипты-обёртки.

Заключение

ИИ-агенты не отменяют инженерную дисциплину — они меняют её форму. Программирование с агентом требует не меньше контроля, чем традиционная разработка, но контроль смещается с чтения кода на проектирование правил и проверок. Если вы продакт без технического бэкграунда, начните с малого: определите чёткие критерии приёмки, настройте CI и требуйте от агента отчёты о проверках. И помните: доверяй, но проверяй — особенно когда код пишет машина.

Источник: https://cms.dolinais.ru/programmirovanie-kak-vesti-proekt-kogda-kod-pishut-ii-agenty