Программирование в 2026: как поддерживать легаси-код и не утонуть в нём

28.11.2026 · PDF-версия статьи с сайта cms.dolinais.ru

Почему легаси снова в центре внимания

Легаси-код — это работающая программа, которую дорого или рискованно менять: нет тестов, документация устарела, авторы ушли, а бизнес зависит от её поведения. В 2026 году ситуация обострилась: ИИ-агенты уверенно генерируют новый код, но старые системы остаются и требуют изменений — добавить интеграцию, закрыть уязвимость, перенести в контейнеры. По данным отраслевых опросов, значительная часть инженерного времени уходит именно на сопровождение существующих систем, а не на greenfield-проекты. Поэтому навык «трогать старое и не ломать» стал таким же важным, как умение писать новое.

В этой статье — практический подход к работе с легаси: как оценить риск, получить страховку в виде тестов, вносить изменения маленькими шагами и использовать ИИ-инструменты там, где они действительно помогают.

Шаг 1. Оцените систему и определите границы

Прежде чем что-то менять, зафиксируйте, что именно вы трогаете. Полезно нарисовать карту зависимостей: какие модули вызывают изменяемый код, какие внешние системы он затрагивает, где хранятся данные.

- **Точки входа.** HTTP-эндпоинты, очереди, cron-задачи, CLI-команды — всё, что может запустить выполнение.
- **Побочные эффекты.** Запись в БД, отправка писем, вызовы платёжных API. Их нельзя тестировать «на живую».
- **Критичность.** Что сломается для пользователя, если модуль упадёт: потеря денег, данных, репутации.

Результат шага — короткий документ на 1–2 страницы и список модулей, которые нельзя менять без тестов. Это не бюрократия, а основа для приоритизации.

Шаг 2. Поставьте «страховку»: тесты-характеристики

Классический совет «покройте код тестами» в легаси не работает: непонятно, какое поведение правильное. На помощь приходят **тесты-характеристики** (characterization tests) — тесты, которые фиксируют текущее поведение системы, даже если оно кажется странным. Их цель — обнаружить изменение, а не доказать корректность.

1. Выберите самый рискованный модуль, который планируете менять.
2. Запустите его на реальных или реалистичных входных данных и запишите результат.
3. Оформите записанный результат как утверждение в тесте (snapshot, golden master).
4. Повторяйте, пока не покроете основные ветки выполнения.

Важно: такие тесты не должны лезть в настоящую БД или внешние API. Используйте тестовые дублёры (stubs, fakes) и фиксируйте контракты. Если поведение системы зависит от времени или случайности — подмените

источники времени и генераторы случайных чисел.

> Правило: перед первым изменением в легаси-модуле у вас должен быть хотя бы один тест, который падает, если вы случайно сломаете ключевой сценарий.

Шаг 3. Меняйте маленькими шагами

Большие переписывания опасны: вы надолго теряете обратную связь и накапливаете риск. В 2026 году практика подтверждает старую истину: маленькие обратимые изменения надёжнее. Работайте по циклу:

- Сформулируйте одно изменение (например, «вынести валидацию в отдельную функцию»).
- Добавьте или обновите тест, который фиксирует новое поведение.
- Внесите изменение, прогоните тесты, выкатите в прод за флагом (feature flag), если это возможно.
- Убедитесь, что метрики не ухудшились, и только потом переходите к следующему шагу.

Флаг функции (feature flag) — переключатель, который позволяет включить новое поведение для части пользователей или выключить его без отката кода. Это дешёвая страховка при работе с легаси.

Шаг 4. Изолируйте и постепенно заменяйте

Если модуль слишком запутан, не пытайтесь переписать его целиком. Сначала выделите его за интерфейс (фасад), чтобы остальная система зависела от абстракции, а не от конкретной реализации. Затем можно заменить реализацию по частям — этот приём известен как «душить» (strangler fig pattern): новая логика постепенно перехватывает вызовы у старой.

Практические шаги:

1. Опишите контракт: какие входы, выходы и ошибки ожидает вызывающий код.
2. Введите адаптер, который пока делегирует вызовы старому коду.
3. Напишите новую реализацию и переключите адаптер на неё для части сценариев.
4. Сравнивайте результаты старой и новой реализации на теневом трафике, прежде чем переключать всех.

Такой подход снижает риск и позволяет остановиться на любом этапе.

Шаг 5. Используйте ИИ-инструменты с умом

В 2026 году ИИ-ассистенты стали обычным инструментом разработчика. Они полезны в легаси, но требуют дисциплины:

- **Объяснение кода.** Попросите агента описать, что делает функция, и проверьте его вывод на небольшом примере.
- **Генерация тестов-характеристик.** Агент может предложить набор входных данных, но утверждения должны быть проверены вами.
- **Рефакторинг.** Не принимайте большие диффы без просмотра: ИИ склонен менять поведение в мелочах.

Ключевое правило: ИИ ускоряет рутину, но ответственность за поведение системы остаётся на инженере.

Перед применением сгенерированного кода прогоняйте тесты и проверяйте граничные случаи.

Что запомнить

- Легаси — это риск, а не только «плохой код». Сначала оцените границы и критичность.
- Тесты-характеристики фиксируют текущее поведение и дают страховку для изменений.
- Маленькие шаги, флаги функций и постепенная замена снижают вероятность крупной аварии.
- ИИ-ассистенты полезны, но требуют проверки: они не знают контекст вашей системы.

Главный вывод для 2026 года: умение безопасно менять старый код — конкурентное преимущество команды. Оно позволяет выпускать новое, не останавливая бизнес и не накапливая технический долг до состояния, когда система становится неприкосновенной.

Источник: https://cms.dolinais.ru/programmirovanie-v-2026-kak-podderzhivat-legasi-kod-i-ne-utonut-v-nem