

Программирование как инженерная дисциплина: от задачи до поддерживаемого кода

21.10.2026 · PDF-версия статьи с сайта cms.dolnais.ru

В 2026 году спорят о языках, агентах и «вайб-кодинге», но суть программирования не меняется: это процесс создания и модификации компьютерных программ, в котором алгоритмы соединяются со структурами данных. Формула Никлауса Вирта «Программы = алгоритмы + структуры данных» остаётся рабочей. Ниже — инженерный взгляд на программирование: как превратить задачу в работающий продукт и не утонуть в сопровождении.

Что такое программирование в инженерном смысле

Программирование — деятельность по проектированию, написанию, проверке и изменению программ. В неё входят не только строки кода, но и требования, архитектурные решения, тесты, документация и эксплуатация. Язык программирования — инструмент; визуальное или zero-code-программирование тоже остаётся программированием, просто с другим уровнем абстракции.

Полезно разделять два режима работы:

- **Решение задачи** — получить корректный результат здесь и сейчас.
- **Создание системы** — сделать так, чтобы результат воспроизводился, проверялся и менялся без разрушения.

Большинство проблем в проектах возникает, когда второй режим подменяют первым.

Шаг 1. Зафиксировать требования и критерии готовности

Начинайте не с кода, а с формулировки: какую проблему решаем, для кого, что считается успехом. Хорошее требование проверяемо. Например, не «ускорить поиск», а «поиск по каталогу отвечает за 300 мс на 95-м проценте запросов при 50 тыс. товаров».

1. Опишите пользовательский сценарий в одном абзаце.
2. Выпишите входные и выходные данные, ограничения и ошибки.
3. Согласуйте критерии приёмки — они станут основой тестов.

Если требование нельзя проверить, его нельзя и реализовать предсказуемо. Это правило экономит недели переделок.

Шаг 2. Выбрать структуры данных и границы модулей

Алгоритмы редко бывают главной проблемой; чаще подводит неудачная модель данных. Прежде чем писать код, ответьте:

- Какие сущности есть в предметной области и как они связаны?
- Где проходит граница модуля или сервиса?

- Какие данные нужно хранить, а какие можно вычислять?

Пример: в задаче учёта заявок сущности «заявка», «исполнитель», «статус» и «история изменений» лучше разделить. Тогда отчётность и уведомления не будут встраиваться в код создания заявки. Границы модулей стоит проверять по числу причин для изменения: если модуль меняется из-за трёх разных отделов, его, вероятно, пора разделить.

Шаг 3. Писать код, который можно проверить

Практические приёмы, которые работают в 2026 году так же, как и раньше:

1. **Маленькие функции с явными входами и выходами.** Функция, которая читает глобальное состояние, сложно тестируется и непредсказуемо ведёт себя при изменении окружения.
2. **Явная обработка ошибок.** Определите, какие ошибки ожидаемы (валидация, отсутствие записи), а какие — сбой (недоступность БД). Смешивать их в одном обработчике опасно.
3. **Логирование по делу.** Фиксируйте идентификатор запроса, ключевые шаги и результат; не логируйте персональные данные и секреты.
4. **Конфигурация отдельно от кода.** Адреса, таймауты и флаги — в переменных окружения или конфигурационных файлах.

Для проверки используйте пирамиду тестов: много быстрых модульных, меньше интеграционных, ещё меньше сквозных. Тест должен падать по понятной причине и не зависеть от порядка запуска.

Шаг 4. Ревью и автоматизация проверок

Код-ревью — просмотр изменений коллегами до слияния. Его цель — не поиск опечаток, а проверка логики, границ ответственности и тестов. Полезный формат: автор описывает, что и зачем изменено, ревьюер проверяет поведение и риски.

- Автоматизируйте форматирование, линтеры и статические проверки — они снимают шум с ревью.
- Ограничивайте размер изменений: небольшой патч проще проверить и откатить.
- Фиксируйте решения в коротких заметках (ADR — architecture decision record), чтобы через полгода не восстанавливать контекст по памяти.

Шаг 5. Эксплуатация: наблюдение, откат, обновления

Программа живёт после релиза. Минимальный набор для сопровождения:

- метрики доступности и времени ответа, журналы ошибок;
- процедура отката релиза и проверенный способ восстановления данных;
- регламент обновления зависимостей и проверки уязвимостей.

Отдельно про ИИ-помощников: в 2026 году они ускоряют генерацию кода и тестов, но ответственность за корректность остаётся на разработчике. Сгенерированный фрагмент нужно прочитать, проверить на тестах и убедиться, что он не тянет лишние зависимости и не раскрывает данные.

Типичные ошибки

- **Преждевременная оптимизация.** Сначала измерьте, потом ускоряйте.
- **Отсутствие тестов на критичные сценарии.** Оплата, авторизация, сохранение данных — в первую очередь.
- **Скрытые зависимости.** Модуль, который сам ходит в чужую БД, сложно менять и тестировать.
- **Документация «потом».** Минимальная инструкция запуска нужна в день первого коммита.

Чек-лист перед сдачей задачи

1. Критерии приёмки выполнены и проверены тестами.
2. Ошибки обработаны, логи не содержат секретов.
3. Изменения прошли ревью, линтеры и сборку.
4. Есть план отката и понятно, как отслеживать результат.
5. Обновлено документация и заметка о принятом решении.

Программирование — это дисциплина принятия решений при неполной информации. Чем раньше вы фиксируете требования, границы и проверки, тем меньше кода придётся переписывать. Язык и инструменты можно сменить; привычка проверять результат и оставлять систему понятной — нет.

Источник: <https://task.dolinais.ru/programmirovanie-kak-inzhenernaya-disciplina-ot-zadachi-do-podderzhivaemogo-koda>