

Программирование: как проверить качество сторонних библиотек перед подключением

28.10.2026 · PDF-версия статьи с сайта cms.dolinalis.ru

Подключить библиотеку — одна строка в манифесте зависимостей. Удалить её через год, когда она перестала обновляться или потянула за собой уязвимость, — отдельный проект. Программирование всё сильнее строится на чужом коде: по данным исследований экосистем, в типичном современном приложении собственный код составляет заметно меньше половины строк, остальное — зависимости. Поэтому проверка библиотеки перед подключением становится такой же инженерной задачей, как проектирование API.

Ниже — практический чек-лист для 2026 года: что смотреть в репозитории, какие метрики собирать и как принять решение, если идеального варианта нет.

Сначала сформулируйте, зачем вам эта зависимость

Зависимость — внешний пакет, код которого ваш проект использует и обновляет вместе с собой. Прежде чем открывать репозиторий, ответьте на три вопроса:

- Какую задачу решает библиотека и нельзя ли закрыть её 50–100 строками своего кода?
- Насколько задача близка к ядру продукта? Если это ядро, сторонняя зависимость повышает риски.
- Что произойдёт, если библиотеку придётся заменить через полгода: сколько мест в коде затронет миграция?

Правило простое: чем ближе зависимость к бизнес-логике, тем строже критерии. Утилита для форматирования дат может быть «какой угодно», а библиотека для работы с платежами или криптографией требует отдельного аудита.

Проверка репозитория: что смотреть в первую очередь

1. Активность и поддержка

Откройте историю коммитов и раздел с релизами. Смотрите не на общее число звёзд, а на даты:

- когда был последний содержательный коммит (не правка README);
- как часто выходят релизы за последние 12 месяцев;
- есть ли ответы мейнтейнеров на свежие issues и pull requests;
- сколько людей имеют права на слияние — проект на одном энтузиасте уязвимее к выгоранию.

Библиотека без релизов больше года — не приговор, но повод проверить, не архивирован ли репозиторий и есть ли форки, которые продолжают работу.

2. Качество кода и тестов

Посмотрите на структуру проекта и CI. Признаки здорового репозитория: настроены линтеры и статический анализ, тесты запускаются на нескольких версиях рантайма, есть покрытие краевых случаев, а не только

«счастливого пути». Загляните в пару файлов исходников: понятные имена, отсутствие закомментированного мусора и гигантских функций говорят о культуре разработки больше, чем описание в README.

3. Безопасность и история уязвимостей

Проверьте пакет в базе уязвимостей (например, OSV или GitHub Advisory) и посмотрите, как проект реагировал на прошлые инциденты. Быстрый патч и публичное раскрытие — хороший знак. Молчание месяцами после сообщения об уязвимости — плохой. Отдельно уточните, включён ли в проекте процесс ответственного раскрытия (security policy).

4. Лицензия

Лицензия определяет, что вы вправе делать с кодом. Разрешительные (MIT, Apache 2.0, BSD) обычно совместимы с коммерческими продуктами. Копилефт-лицензии (GPL, AGPL) требуют раскрытия производного кода при распространении или сетевом использовании — для закрытого продукта это критично. Проверяйте лицензию не только у самой библиотеки, но и у её транзитивных зависимостей: инструменты вроде лицензионных сканеров показывают полное дерево.

5. Вес и транзитивные зависимости

Одна библиотека может притащить десятки пакетов. Посчитайте размер дерева зависимостей и оцените, не тянет ли маленькая утилита за собой целый фреймворк. Для фронтенда это напрямую влияет на размер бандла и скорость загрузки, для бэкенда — на время установки и поверхность атаки.

Как проверить библиотеку на практике: пошаговый порядок

1. **Соберите шорт-лист** из 2–3 кандидатов, а не берите первый результат поиска.
2. **Прогоните каждый по критериям выше** и зафиксируйте выводы в короткой таблице: дата релиза, число мейнтейнеров, лицензия, известные уязвимости, размер дерева.
3. **Сделайте пробный прототип**: подключите библиотеку в отдельной ветке и напишите небольшой сценарий, близкий к вашей задаче. Так вы оцените API и документацию на реальном примере.
4. **Проверьте миграционный путь**: насколько легко обернуть зависимость в собственный адаптер, чтобы замена не затронула весь код.
5. **Зафиксируйте версию** и настройте автоматическое отслеживание обновлений и уязвимостей в CI.

Хорошая зависимость — та, которую легко заменить. Изолируйте сторонний код за собственным интерфейсом, и решение о выборе библиотеки перестанет быть необратимым.

Когда стоит форкнуть или написать своё

Форк оправдан, если библиотека закрывает 90% задачи, но Entwicklung остановлена, а альтернатив нет. Перед этим ответьте на пять вопросов: кто будет поддерживать форк, сколько времени займёт синхронизация с апстримом, готовы ли вы публиковать патчи, не нарушает ли форк лицензию и что произойдёт, если апстрим «оживёт» и уйдёт вперёд. Если хотя бы на два вопроса нет внятного ответа, своя реализация узкой функции часто дешевле.

Что настроить после подключения

Выбор библиотеки — не разовое действие, а процесс. В 2026 году базовый минимум выглядит так: автоматическое сканирование зависимостей в CI, политика обновления (например, патчи — сразу, минорные версии — раз в спринт), фиксация версий через lock-файл и регулярный обзор дерева зависимостей раз в квартал. Такой подход превращает работу с чужим кодом из лотереи в управляемый инженерный процесс.

Источник: <https://cms.dolnais.ru/programmirovanie-kak-proverit-kachestvo-storonnih-bibliotek-pered-podklyucheniem>