Вайбкодинг в 1С и Битрикс24

Практическое руководство для разработчиков

Как передать часть работы ИИ-помощнику и не получить взамен сломанную базу и конфигурацию, снятую с поддержки.

La Chatte · LC STUDIO · Новосибирск
la-chatte.com · info@la-chatte.com · +7 383 380-81-89

Содержание

Глава 1. Что такое вайбкодинг и чем он не является

Коротко

Вайбкодинг — это работа, при которой основную часть кода печатает ИИ-помощник, а программу задаёт, проверяет и отвечает за неё человек. Разработчик описывает словами, что нужно сделать. Помощник пишет код. Разработчик читает результат, находит ошибки и правит.

Слово появилось в феврале 2025 года: его пустил в оборот Андрей Карпатый, один из основателей OpenAI, в короткой заметке о том, как он собирает мелкие программы, почти не набирая код руками. Термин прижился быстрее, чем успело сложиться понимание, что он означает.

Чем вайбкодинг не является

Здесь больше всего путаницы, поэтому начнём с того, чем он не является.

- Это не «нейросеть напишет программу за вас». Помощник пишет фрагменты. Собрать из них работающий продукт, проверить и запустить — работа человека.
- Это не программирование без знаний. Чтобы найти ошибку в чужом коде, надо уметь этот код читать. Помощник ошибается регулярно, и без проверки его ошибки уезжают в работающую систему.
- Это не замена разработчиков. Меняется распределение времени внутри работы, а не потребность в человеке.
- Это не автодополнение. Подсказка следующей строки существует давно. Вайбкодинг — про другое: вы описываете задачу целиком и получаете целиком решение, которое надо разобрать.

Что меняется на практике

Меняется самое дорогое в разработке — часы человека.

В контролируемом эксперименте GitHub на 95 разработчиках типовая задача с ИИ-помощником закрывалась за 1 час 11 минут вместо 2 часов 41 минуты — почти вдвое быстрее (GitHub Research).

По оценке McKinsey, на понятных и небольших задачах генерации нового кода помощник экономит около половины времени разработчика. На сложных и непривычных задачах выигрыш падает до 10% и меньше.

Из этих двух цифр следует главный практический вывод: вайбкодинг сильно помогает на простом и почти не помогает на сложном. Поэтому решение о том, какую задачу ему отдать, важнее умения формулировать запрос.

Что стало окупаться

Раньше многие небольшие задачи не делали не потому, что они трудные, а потому что ставить на них разработчика было невыгодно. Медиана зарплаты backend-разработчика в России в 2026 году — 251 тыс. ☐ в месяц, около 1 500 ☐ за рабочий час (Хабр Карьера, 2026). Неделя работы на внутренний калькулятор — это примерно 60 тыс. ☐, и такие задачи годами лежали в очереди.

Когда та же работа занимает день, очередь начинает разбираться. Обычно первыми делают вот это:

- внутренний калькулятор или конвертер, которым пользуется один отдел;
- панель с цифрами для руководителя вместо ручной выгрузки в таблицу;
- разовый отчёт, который просили полгода;
- мелкая доработка старой системы, до которой не доходили руки;
- скрипт, переносящий данные из одного места в другое.

Кто за это отвечает

Отдельно и прямо, потому что это главный источник проблем.

Помощник не несёт ответственности за код. Если написанное им уронило продажи или испортило данные, объяснять это будет человек, который нажал кнопку. По опросу Stack Overflow за 2025 год 66% разработчиков называют главной проблемой ИИ-кода то, что он «почти правильный, но не совсем» — и тратят на проверку больше времени, чем рассчитывали.

«Почти правильный» — самое опасное состояние кода. Явно сломанный код падает сразу и его чинят. Почти правильный работает, проходит беглый просмотр и ломается через месяц на данных, которых никто не ждал.

Что дальше

В следующей главе — кому вайбкодинг подходит, а кому навредит. Это решается до выбора инструментов, и ошибка здесь обходится дороже всех остальных.

Глава 2. Кому подходит, а кому навредит

Коротко

Вайбкодинг помогает тому, кто умеет читать код. Он вредит тому, кто читать код не умеет, — потому что такой человек не заметит ошибку и узнает о ней, когда она уже что-то сломала.

Это единственное деление, которое имеет значение. Не язык, не опыт в годах, не размер компании.

Кому подходит

- Разработчику с опытом. Самый выигрышный случай. Вы и так знаете, каким должен быть результат, и быстро видите, где помощник соврал. Экономия времени максимальная, риск минимальный.
- Разработчику на незнакомой платформе. Вы умеете программировать, но пришли в новую для себя систему. Помощник ускоряет вход в неё в разы — при условии, что вы проверяете каждый шаг по документации.
- Аналитику, который пишет запросы к данным. Отчёты, выгрузки, обработка таблиц. Ошибка видна сразу по результату, цена ошибки невысокая.
- Технически грамотному владельцу небольшого бизнеса. Если вы понимаете, как устроены ваши данные, и готовы разбираться — соберёте себе рабочий инструмент. С оговорками из следующего раздела.

Кому навредит

Здесь придётся быть неприятно прямым.

- Тому, кто не программирует совсем. Обещание «соберите приложение без знаний» продаётся хорошо, а заканчивается одинаково: получается то, что запускается и выглядит рабочим, но содержит ошибки, которые человек не в состоянии увидеть. Пока это личный проект — не страшно. Когда там деньги или чужие персональные данные — страшно.
- Тому, кто спешит и не проверяет. Вайбкодинг ускоряет и написание, и накопление ошибок. Без проверки вы получаете не быструю разработку, а быстрый технический долг.
- Тому, кто работает с нагруженной или критичной системой. Расчёт зарплаты, склад, касса, медицинские данные. Здесь цена «почти правильного» кода несопоставима с экономией часа.
- Команде без код-ревью. Если код никто не смотрит перед тем, как он уходит в работу, вайбкодинг просто увеличит поток непроверенного кода.

Как понять, что задача подходит

Четыре вопроса. Если на все четыре ответ «да» — отдавайте помощнику.

1. Я могу описать словами, что должно получиться? Не «сделай красиво», а «на вход список заказов, на выход таблица с суммой по каждому клиенту за месяц».
2. Я пойму, что результат неправильный? Если проверить нечем, вы не узнаете об ошибке.
3. Что случится, если это сломается? Отчёт покажет неверную цифру — переживём. Клиенту уйдёт неправильный счёт — уже нет.
4. Задача обычная или редкая? На типовых вещах помощник силён. На редких и специфичных для вашей системы — слаб, и это подтверждается оценкой McKinsey: на непривычных задачах выигрыш падает до 10% и меньше.

Правило одного шага

Начинать стоит не с продукта, а с задачи, которая не страшно если не получится.

Внутренний отчёт, который сейчас собирают руками. Скрипт, который переносит данные. Черновик формы. Сделайте одну такую вещь целиком: поставьте задачу, получите код, проверьте, запустите. После этого станет понятно, сколько времени уходит именно у вас — а это единственная цифра, которая имеет значение для решения.

Что дальше

Следующая глава — про рабочее место: чем пользоваться, какие доступы нужны и что нельзя показывать помощнику.

Глава 3. Рабочее место: инструменты и доступы

Коротко

Рабочее место для вайбкодинга — это три решения: чем пользоваться, куда пускать помощника и что ему нельзя показывать. Третье важнее первых двух, но про него обычно не думают.

Три вида инструментов

Все помощники для кода делятся на три группы по тому, сколько они видят.

- Чат в браузере. Вы вставляете кусок кода и вопрос, получаете ответ. Видит только то, что вы вставили. Ничего не устанавливается, начать можно за минуту. Подходит, чтобы попробовать и разобрать отдельный фрагмент.
- Помощник внутри редактора кода. Видит открытый файл и соседние. Понимает, как устроен ваш проект, и предлагает правки на месте. Основной рабочий инструмент для большинства задач.
- Агент, который сам правит файлы. Получает задачу, сам открывает нужные файлы, вносит изменения, запускает проверки. Экономит больше всего времени и требует больше всего внимания: он меняет то, о чём вы не просили, если задача описана неточно.

Начинать разумно со второго. Первый слишком мелкий для настоящей работы, третий требует привычки проверять каждое изменение.

Что нельзя отдавать наружу

Главная глава этой книги для тех, кто работает в компании, а не над личным проектом.

Когда вы вставляете код в чат или подключаете помощника к проекту, содержимое уходит на чужой сервер. Дальше зависит от условий сервиса: где-то данные удаляют, где-то хранят, где-то используют для обучения. Разбираться в этом надо до первого запроса, а не после.

Что не показывают помощнику никогда:

- Пароли, ключи доступа и токены. Даже в примере, даже «на минуточку». Ключ, попавший в чужой лог, считается скомпрометированным — его надо менять.
- Персональные данные людей. Реальные фамилии, телефоны, адреса, номера документов, медицинские сведения. Для примеров есть выдуманные данные.
- Коммерческую тайну. Формулы расчёта цен, условия договоров с поставщиками, клиентские базы.

- Код, закрытый соглашением о неразглашении. Если вы подписывали NDA, отправка кода наружу — его нарушение, независимо от намерений.

Как работать, если показывать нельзя

Три рабочих способа, от простого к сложному.

1. Обезличить. Замените реальные данные выдуманными, названия полей оставьте. Помощнику для написания кода нужна структура, а не содержимое.
2. Спросить про кусок, а не про систему. Вместо «вот наш модуль расчёта скидок» — «как отсортировать список по двум полям». Задача решается, тайна остаётся.
3. Поставить помощника внутри компании. Модель работает на вашем сервере, наружу не уходит ничего. Дороже и требует администрирования, но для банков, медицины и госсектора часто единственный допустимый вариант.

Что настроить до первой задачи

Полчаса, которые потом экономят дни.

- Система контроля версий. Обязательна. Помощник иногда меняет то, что вы не просили. Без возможности откатиться вы будете восстанавливать файлы руками.
- Отдельная ветка для его правок. Не работайте помощником сразу в основной ветке.
- Автоматические проверки. Даже простейшие. Если код запускается и базовые проверки проходят, часть ошибок отсеется без вас.
- Правила проекта в отдельном файле. Многие помощники читают файл с описанием проекта: какой стиль кода принят, чего делать нельзя, куда не лезть. Один раз написали — перестали повторять в каждой задаче.

Чего не надо делать на старте

Не покупайте пять инструментов сразу, чтобы сравнить. Возьмите один и проработайте им месяц. Разница между инструментами меньше, чем разница между «умею ставить задачу» и «не умею» — а это следующая глава.

Глава 4. Как ставить задачу, чтобы код был рабочим

Коротко

Помощник пишет плохой код, когда задача поставлена плохо. Это самая частая причина разочарования, и она чинится не сменой инструмента, а привычкой описывать задачу целиком.

Хорошо поставленная задача содержит четыре вещи: что на входе, что должно получиться, чего делать нельзя, как проверить результат.

Четыре части задачи

1. Что на входе. Откуда берутся данные, как они устроены, что в них может быть не так. Пустые поля, пропуски, повторы — про это надо сказать заранее, иначе помощник напишет код для идеальных данных.
2. Что должно получиться. Не «отчёт», а какие именно колонки, в каком порядке, в каком виде, куда сохранить.
3. Чего делать нельзя. Не трогать соседние файлы, не менять структуру базы, не подключать сторонние библиотеки, не переписывать то, что работает.
4. Как проверить. На каких данных прогнать и что должно получиться. Если проверить нечем — вы не узнаете об ошибке.

Плохо и хорошо

Плохо: «Сделай выгрузку заказов».

Помощник не знает, откуда брать заказы, за какой период, какие поля нужны и куда класть результат. Он всё это додумает — и почти наверняка не так.

Хорошо: «В таблице заказов есть поля: номер, дата, клиент, сумма, статус. Нужен файл с заказами за прошлый месяц в статусе „оплачен“. Колонки: клиент, количество заказов, общая сумма. Отсортировать по сумме, по убыванию. У части заказов клиент не заполнен — такие собрать в отдельную строку „без клиента“. Не меняй саму таблицу заказов, только читай».

Длиннее в десять раз. Экономит час.

Одна задача за раз

Соблазн описать всё сразу и получить готовый продукт — самая дорогая ошибка новичка.

Помощник ответит на большую задачу большим куском кода. Проверить его целиком трудно, а ошибка внутри будет спрятана в объёме. Когда что-то не работает, непонятно, какая из десяти частей виновата.

Разбивайте. Сделали шаг — проверили — зафиксировали в системе контроля версий — следующий шаг. Медленнее ощущается, быстрее получается.

Как разговаривать, когда не получилось

Помощник ошибся. Дальше есть два пути, и один из них тупиковый.

Тупиковый: «не работает», «всё ещё не работает», «ты неправильно сделал». Помощник не видит вашего экрана и не знает, что именно сломалось. В ответ он будет переписывать код наугад, каждый раз по-новому, и вы уйдёте дальше от рабочего варианта, чем были.

Рабочий: скажите, что именно произошло. Какую ошибку выдало, на каких данных, что ожидали получить и что получили. Тогда исправление будет точечным.

Если после двух-трёх попыток не сдвинулось — остановитесь. Начните новую задачу с чистого листа и опишите её заново, подробнее. Это почти всегда быстрее, чем спорить дальше.

Правила проекта вместо повторов

Если вы каждый раз пишете «у нас отступы в четыре пробела», «не подключай сторонние библиотеки», «комментарии по-русски» — вынесите это в файл с правилами проекта. Помощник прочитает его сам, и задача сократится до сути.

Чего не ждать

Хорошая постановка задачи повышает долю рабочего кода, но не доводит её до сотни. Проверять всё равно придётся — про это следующая глава.

Глава 5. Вайбкодинг в 1С: встроенный язык и запросы

Коротко

Помощник на 1С ошибается заметно чаще, чем на популярных языках. Не потому что 1С сложнее, а потому что кода на 1С в открытом доступе мало. Модели учились в основном на английском коде из открытых репозиториев, а разработка на 1С почти вся закрытая и русскоязычная.

Практический вывод простой: на 1С вайбкодинг работает, но доля кода, который надо переделывать, выше. Проверять надо строже, а задачи давать мельче.

Почему на 1С труднее

- Два разных языка в одной задаче. Встроенный язык и язык запросов устроены по-разному. Помощник их путает: пишет в запросе то, что работает только в коде модуля.
- Язык запросов похож на SQL, но это не SQL. Сходство обманывает. Помощник тянет привычные конструкции из SQL, которых в 1С нет.
- Клиент и сервер разделены. Директивы компиляции определяют, где выполняется процедура. Обращение к базе из клиентского контекста — классическая ошибка, и помощник допускает её постоянно, потому что в других языках такого разделения нет.
- Русские имена. Встроенный язык русскоязычный. Помощник иногда сползает на английские синонимы или смешивает их в одном модуле.
- Версии платформы. То, что появилось в свежих версиях, помощник может не знать. Обратное тоже бывает: предлагает давно устаревшее.

Главная беда: выдуманные методы

Это самая частая и самая опасная ошибка на 1С. Помощник вызывает метод, которого в платформе не существует. Имя выглядит правдоподобно, стиль совпадает с настоящими методами, документация к нему «как будто есть».

Такой код не запускается, и это хорошая новость: ошибка вылезает сразу. Плохая новость в том, что на исправление уходит время, а помощник, которого попросили починить, часто предлагает другой несуществующий метод.

Что делать: держать рядом синтакс-помощник платформы и проверять по нему каждое незнакомое имя. Это быстрее, чем спорить с помощником.

Пять ошибок, которые помощник делает на 1С регулярно

Это не список всех возможных ошибок, а те, что повторяются чаще всего. Узнав их в лицо, вы будете ловить большую часть проблем ещё при чтении кода.

1. SQL вместо языка запросов 1С

Самая частая. Помощник видит, что вы просите запрос, и пишет на SQL — потому что SQL он знает несравнимо лучше.

Как выглядит: в тексте запроса появляются SELECT, FROM, WHERE, LEFT JOIN, GROUP BY. В 1С это ВЫБРАТЬ, ИЗ, ГДЕ, ЛЕВОЕ СОЕДИНЕНИЕ, СГРУППИРОВАТЬ ПО.

Иногда получается смесь: половина конструкций правильная, половина нет. Такое читается особенно плохо, потому что глаз цепляется за знакомое и пропускает чужое.

2. Конструкции, которых в языке запросов нет

Отдельный случай предыдущей ошибки, но коварнее: помощник берёт функцию из SQL, для которой в 1С есть аналог с другим именем.

- LIMIT или TOP — в 1С это ВЫБРАТЬ ПЕРВЫЕ 10.
- CASE WHEN ... THEN ... ELSE ... END — в 1С ВЫБОР КОГДА ... ТОГДА ... ИНАЧЕ ... КОНЕЦ.
- ISNULL или COALESCE — в 1С ЕСТЬNULL.
- SUBSTRING — в 1С ПОДСТРОКА.
- ORDER BY — в 1С УПОРЯДОЧИТЬ ПО.

Ошибка вылезает при проверке запроса, так что до данных не доходит. Но время съедает: помощник, которого попросили исправить, нередко заменяет одну несуществующую конструкцию на другую.

3. Обращение к базе из клиентского контекста

Классика, которую помощник допускает постоянно, потому что в других языках разделения на клиент и сервер просто нет.

Как выглядит: процедура помечена &НаКлиенте, а внутри обращение к Справочники, Документы или выполнение запроса. Так нельзя: эти объекты доступны только на сервере.

Обратная ошибка тоже встречается: в процедуре &НаСервере помощник вызывает диалог или показывает предупреждение пользователю. На сервере интерфейса нет.

Как ловить: при чтении кода первым делом смотреть на директиву над процедурой и сверять с тем, что внутри.

4. Запрос внутри цикла

Помощник пишет понятный человеку код: перебрать строки таблицы, для каждой сходить в базу за данными. Работает. На десяти строках.

На десяти тысячах это десять тысяч обращений к базе вместо одного запроса. Система не падает — она просто становится очень медленной, и виноватым назначают «1С тормозит».

Как ловить: искать обращения к базе внутри Для Каждого. Правильный путь — один запрос, который сразу вернёт всё нужное.

5. Выдуманные методы

Помощник вызывает метод, которого у объекта нет. Имя правдоподобное, стиль совпадает с настоящими методами платформы, выглядит убедительно.

Такой код не запускается, и это спасает: ошибка вылезает сразу. Опасность в другом — на исправление уходит время, а помощник в ответ часто предлагает следующий несуществующий метод.

Как ловить: держать открытым синтакс-помощник платформы и проверять по нему каждое незнакомое имя. Это занимает секунды и надёжнее любого спора с помощником.

Если вы собираете такие случаи у себя — записывайте их. Разборов реальных ошибок помощника на 1С в открытом доступе почти нет, и это самое ценное, что можно опубликовать по теме.

Что отдавать помощнику на 1С

По убыванию отдачи.

- Запросы к данным. Собрать выборку по условиям, соединить таблицы, сгруппировать. Результат виден сразу, ошибка ловится проверкой на реальных данных.
- Обработки для разовых задач. Перенести данные, найти расхождения, массово поправить реквизит. Живут один раз, цена ошибки ограничена.
- Разбор чужого кода. Объяснить, что делает модуль, доставшийся от предыдущего разработчика. Здесь помощник хорош даже на 1С: понимать чужое ему проще, чем писать новое.
- Черновик печатной формы или отчёта. Структуру набросает, оформление доводите руками.
- Комментарии и описание к готовому коду. Скучная работа, которую никто не любит.

Что не отдавать

- Расчёты, где важна каждая копейка. Зарплата, налоги, себестоимость, взаиморасчёты.

- Проведение документов и движения по регистрам. Ошибка здесь портит учётные данные, и обнаруживается это при закрытии месяца.
- Всё, что меняет структуру конфигурации. Про это следующая глава.
- Код, который должен выдержать нагрузку. Помощник напишет запрос в цикле, и на тысяче строк это ляжет.

Как ставить задачу на 1С

К четырём частям из главы про постановку задачи добавляются три вещи, специфичные для платформы.

1. Назовите версию платформы и конфигурацию. «1С:Предприятие 8.3, Управление торговлей» — иначе получите усреднённый ответ.
2. Скажите, где должен выполняться код. Модуль объекта, общий модуль, модуль формы, клиент или сервер. Без этого помощник выберет наугад.
3. Опишите метаданные, которые участвуют. Имена справочников, документов, регистров и их реквизитов. Помощник не видит вашу конфигурацию и будет выдумывать имена, если их не дать.

Проверка на 1С: что смотреть отдельно

Помимо общего порядка из главы про проверку.

- Запросы в цикле. Самая частая проблема с производительностью. Работает на десяти записях, кладёт базу на десяти тысячах.
- Обращение к базе из клиента. Проверьте директивы компиляции.
- Работа с датами. Границы периода, конец дня, часовые пояса.
- Пустые ссылки. Помощник обычно пишет код так, будто реквизит всегда заполнен.
- Права доступа. Код, который работает у администратора и падает у кладовщика.

Что дальше

Следующая глава — про обмен через OData и про то, как дорабатывать типовую конфигурацию, не потеряв обновления.

Глава 6. 1С: обмен через OData и типовые конфигурации

Коротко

Две задачи, которые чаще всего приносят в 1С со стороны: забрать данные наружу и доработать типовую конфигурацию. Обе имеют правильный способ решения и распространённый неправильный. Помощник по умолчанию предлагает неправильный, потому что тот проще.

Обмен через OData

В платформе есть встроенный способ отдать данные наружу — автоматический REST-сервис. Его публикуют на веб-сервере, после чего к справочникам, документам и регистрам можно обращаться обычными запросами по сети. Ничего писать для этого не надо, надо настроить.

Это лучший вариант для интеграции, и вот почему он важнее самописного обмена:

- не надо поддерживать свой код обмена при обновлениях;
- состав данных задаётся настройкой публикации, а не переписыванием;
- права работают штатные — можно отдать наружу отдельного пользователя с ограниченным доступом.

Где помощник обычно ошибается. Он предлагает написать свой обмен файлами или свой веб-сервис, потому что таких примеров в интернете больше. Если задача — «отдать данные наружу», сначала проверьте, закрывается ли она публикацией стандартного сервиса.

Что спрашивать у помощника про OData

Здесь он полезен и надёжен, потому что речь про обычные сетевые запросы, а не про язык 1С.

- Разобрать структуру ответа и объяснить, что в нём.
- Написать код на стороне принимающей системы, который заберёт и разложит данные.
- Составить фильтр выборки, чтобы не тянуть всё.
- Обработать постраничную выдачу при больших объёмах.

Проверять всё равно надо: имена сущностей в сервисе строятся из имён метаданных по правилам платформы, и помощник их иногда угадывает неверно. Точный список всегда можно получить у самого сервиса.

Безопасность обмена

Пункт, который пропускают чаще всего.

- Отдельный пользователь только для обмена, с минимальными правами. Не администратор.
- Доступ по защищённому соединению, не по открытому.
- Доступ ограничен по адресам, откуда приходят запросы.
- Пароль этого пользователя — не в коде и не в чате с помощником.

Типовая конфигурация: главное правило

Типовая конфигурация обновляется поставщиком. Пока она на поддержке, обновления ставятся почти автоматически. Как только вы правите её напрямую, каждое обновление превращается в ручное сравнение и объединение — и так навсегда.

Правильный способ дорабатывать — расширения конфигурации. Доработка живёт отдельным файлом, основная конфигурация остаётся нетронутой и обновляется как обычно.

Расширения закрывают большую часть задач: добавить реквизит, изменить форму, перехватить процедуру, добавить отчёт или печатную форму.

Почему это важно для вайбкодинга

Прямая связь: помощник не знает, что ваша конфигурация типовая и стоит на поддержке. Он предложит поправить модуль напрямую, потому что так короче.

Один такой принятый совет стоит дороже всего сэкономленного времени: вы получаете конфигурацию, снятую с поддержки, и обновления, которые теперь делает человек неделю вместо часа.

Что делать: в задаче помощнику прямо писать, что конфигурация типовая и доработка должна быть расширением. И проверять результат: если он предлагает изменить существующую процедуру, спросите, как то же самое сделать через расширение.

Что обычно выносят в расширение

Порядок примерно такой, от самого частого к редкому.

- Дополнительные реквизиты и сведения. Своё поле у справочника или документа. Закрывается расширением почти всегда.
- Изменения формы. Добавить поле, спрятать лишнее, поменять порядок элементов, дописать проверку при записи.
- Свои отчёты и печатные формы. Полностью новые объекты, к типовым не привязаны.

- Перехват процедур. Дописать поведение до или после типовой процедуры, не трогая её саму.
- Свои роли и права. Когда типовых разграничений не хватает.

Как формулировать задачу, чтобы получить расширение

Помощник по умолчанию предложит правку модуля. Разница в постановке задачи.

Даёт неправильный ответ: «как при проведении расходной накладной проверять остаток на складе».

Даёт правильный: «конфигурация типовая и стоит на поддержке, снимать нельзя. Нужна проверка остатка при проведении расходной накладной. Сделать расширением, через перехват процедуры проведения. Типовой модуль не изменять».

Если помощник всё равно предлагает править существующую процедуру — спросите прямо: «как сделать то же самое расширением, не снимая конфигурацию с поддержки». Обычно после этого он даёт правильный вариант: знания у него есть, он просто идёт по короткому пути, пока его не остановить.

Неочевидное про OData

Три вещи, о которых узнают на практике.

- Имена в сервисе не совпадают с именами в конфигураторе. Они строятся по правилам платформы: у справочников, документов и регистров разные приставки. Помощник эти правила знает приблизительно и часто угадывает неверно. Точный список всегда отдаёт сам сервис — запросите его и работайте по нему, а не по догадкам.
- Выдача постраничная. Первый запрос вернёт часть данных, и если этого не учесть, вы будете думать, что записей мало. Помощник про это забывает почти всегда.
- Скорость. Сервис удобен, но это не способ выкачать миллион строк. Для больших объёмов лучше отбор на стороне 1С, а не выгрузка всего и фильтрация снаружи.

Если у вас есть свой разбор обмена — какая система, какие данные, что пошло не так — он ценнее любого общего описания. Такие тексты почти не публикуют.

Чего расширениями не сделать

Честно о границах, иначе совет выглядит как обещание.

- Глубокая переделка логики проведения документа.
- Изменение структуры хранения данных в типовых объектах.
- Некоторые изменения в модулях, где поставщик не оставил точек расширения.

В этих случаях приходится снимать с поддержки — но это осознанное решение с понятной ценой, а не случайность из-за принятого совета помощника.

Что дальше

Следующая глава — про Битрикс24: там та же развилка между быстрым способом и способом, который переживёт обновление.

Глава 7. Битрикс24: доработки без потери обновлений

Коротко

В Битрикс24 та же развилка, что в 1С: есть способ быстрый и есть способ, который переживёт обновление. Помощник по умолчанию предлагает быстрый.

И ещё одно, что надо понять до первой задачи: облачный Битрикс24 и коробочный — это разные истории. В облаке своего серверного кода нет вообще, там только внешние запросы и приложения. В коробке можно почти всё, и именно там легко сломать обновления.

Что можно в облаке

Если у вас облачный портал, доступны три способа что-то автоматизировать.

- Вебхуки. Самый быстрый вход. Получаете адрес с ключом и обращаетесь к порталу обычными сетевыми запросами: создать сделку, найти контакт, добавить задачу. Помощник с этим справляется хорошо, потому что это обычная работа с сетевым интерфейсом.
- Приложения. Полноценный способ: своя авторизация, встраивание в интерфейс, права. Дольше, но правильно, если решение пойдёт не одному клиенту.
- Роботы и бизнес-процессы. Настраиваются мышкой, кода не требуют. Прежде чем писать код, проверьте, не закрывается ли задача здесь: это самый дешёвый в поддержке вариант.

Что можно в коробке и где ловушка

В коробочной версии есть доступ к файлам, и это одновременно возможность и главный риск.

Правило одно: свой код живёт в отдельной папке для доработок, а не в системных файлах. Всё, что положено в системные папки, будет затёрто при обновлении. Не «может быть затёрто», а будет.

Помощник об этом не знает. Он предложит поправить файл там, где нашёл нужную функцию, — потому что так короче и потому что примеры в интернете часто именно такие, старые.

Что делать: в задаче прямо писать, что доработка должна лежать в папке для локальных изменений и подключаться обработчиком события, а не правкой системного файла.

Обработчики событий вместо правки чужого кода

Основной приём, который делает доработки безопасными. Вместо того чтобы менять существующую функцию, вы подписываетесь на событие: «когда создаётся сделка — сделай вот это».

Плюсы очевидны: системный код не тронут, обновления ставятся, вашу доработку видно отдельно и легко отключить.

Помощнику эту конструкцию надо задавать явно. На вопрос «как сделать так, чтобы при создании сделки происходило Х» он с равной вероятностью предложит и обработчик события, и правку системного файла.

Как отличить хороший совет от плохого

Три ситуации, в которых помощник почти наверняка предложит короткий путь вместо правильного.

Ситуация 1. «Нужно, чтобы при создании сделки уходило уведомление». Короткий путь: найти файл, где обрабатывается создание сделки, и дописать туда вызов. Переживёт до первого обновления. Правильный: подписаться на событие создания сделки, обработчик положить в папку для локальных доработок. Системный код не тронут.

Ситуация 2. «Надо изменить поведение стандартного компонента». Короткий путь: отредактировать сам компонент. Это гарантированная потеря правок при обновлении. Правильный: скопировать компонент в свою папку и дорабатывать копию, либо обойтись параметрами и шаблоном. Оригинал остаётся оригиналом.

Ситуация 3. «Нужно свести данные с внешней системой». Короткий путь: писать напрямую в таблицы базы. Работает быстро и ломает целостность данных: платформа не узнает об изменениях, кеши останутся старыми. Правильный: только через штатный программный интерфейс. Медленнее, зато данные остаются согласованными.

Один вопрос, который проверяет любой совет

Прежде чем принять код, спросите себя: что произойдёт с этой правкой при обновлении платформы?

Останется на месте — хорошо. Будет затёрта — переделывайте. Не знаете — значит правка в системном файле, и ответ «будет затёрта».

Тот же вопрос можно задать прямо помощнику: «переживёт ли это обновление и где именно лежит файл». Отвечает он честно — просто сам по себе об этом не думает.

Свои случаи из практики стоит собирать отдельно: где помощник предложил править системный файл и чем это закончилось. Такие разборы читают внимательнее любых общих рекомендаций.

Где помощник ошибается на Битрикс24

- Устаревшие примеры. Платформа старая, в интернете много кода десятилетней давности. Помощник его знает и предлагает.
- Путает облако и коробку. Даёт серверный код тому, у кого облако, где выполнять его негде.
- Выдуманные методы интерфейса. Как и в 1С: имя правдоподобное, метода нет. Проверяется по документации разработчика.
- Игнорирует ограничения на частоту запросов. Код работает на десяти записях и упирается в лимит на тысяче.
- Не думает о правах. Вебхук выполняется от имени создавшего его пользователя. Помощник напишет код, который увидит больше, чем должен видеть конечный пользователь.

Безопасность вебхуков

Короткий, но обязательный раздел.

- Ключ вебхука — это пароль. Попал в чат с помощником, в репозиторий или в переписку — надо отзывать и создавать заново.
- Создавайте вебхук с минимальным набором прав, а не «на всё».
- Для интеграции лучше отдельный служебный пользователь, а не аккаунт руководителя.

Порядок выбора решения

Прежде чем писать код, пройдите сверху вниз и остановитесь на первом подходящем.

1. Закрывается роботом или бизнес-процессом? Настройте, кода не надо.
2. Нужен обмен с внешней системой? Вебхук.
3. Решение пойдёт нескольким клиентам или встраивается в интерфейс? Приложение.
4. Коробка и нужна своя логика внутри? Обработчик события в папке для доработок.
5. Правка системного файла? Не делайте. Если кажется, что иначе никак, — вернитесь к пункту 4 и опишите задачу подробнее.

Что дальше

Дальше — общие главы про проверку кода и про то, где вайбкодинг не работает. Они применимы к обеим платформам и читаются после этой.

Глава 8. Как проверять код, который написал помощник

Коротко

Проверка — это не последний этап вайбкодинга, а его основная часть. Код, который написал помощник, чаще всего выглядит правильным. Проблема именно в этом.

По опросу Stack Overflow за 2025 год 66% разработчиков называют главной трудностью ИИ-кода то, что он «почти правильный, но не совсем». Почти правильный код проходит беглый просмотр, запускается и ломается позже — на данных, которых не ждали.

Пять типичных ошибок помощника

1. Выдуманные функции и методы. Помощник уверенно вызывает то, чего в вашей системе не существует. Выглядит правдоподобно, называется логично, но такого нет. Ловится запуском и проверкой по документации.
2. Код для идеальных данных. Работает на аккуратном примере и падает на реальных: пустое поле, пропуск, повтор, неожиданный формат даты.
3. Молчаливое проглатывание ошибок. Помощник любит обернуть опасное место так, чтобы оно не падало. В результате ошибка происходит, но никто о ней не узнаёт, а данные тихо портятся.
4. Правки не там, где просили. Особенно у агентов, которые сами открывают файлы. Попросили поменять одно — заодно переписано соседнее, которое работало.
5. Устаревший способ. Помощник предлагает то, что было правильным несколько лет назад. Код рабочий, но по нынешним меркам плохой или небезопасный.

Порядок проверки

Пять шагов, по возрастанию затрат времени. Первые два обязательны всегда.

1. Прочитать целиком. Не по диагонали. Если в коде есть места, которые вы не понимаете, — это не «сложно написано», это места, где вы не заметите ошибку. Просите объяснить или переписать проще.
2. Посмотреть, что именно изменилось. Сравнение с прошлой версией показывает все правки, включая те, о которых не просили. Здесь ловятся правки не там, где надо.
3. Запустить на настоящих данных. Не на примере из задачи. На тех, что реально приходят, со всей их грязью.
4. Проверить края. Пустой список, ноль, отрицательное число, очень длинная строка, повторяющаяся запись, дата 29 февраля. Здесь ломается чаще всего.
5. Дать посмотреть человеку. Для всего, что уходит в работу. Свой код после долгого сидения над ним читается плохо.

На что смотреть отдельно

- Работа с деньгами. Округление, копейки, валюты. Ошибка на копейку в отчёте за год превращается в расхождение, которое потом ищут неделю.
- Удаление и перезапись. Любое место, где что-то стирается или заменяется, читается дважды.
- Права доступа. Помощник по умолчанию не думает о том, кому что можно видеть.
- Запросы к данным без ограничений. Работает на сотне записей, кладёт систему на миллионе.

Приём: попросить объяснить

Прежде чем принимать код, попросите помощника объяснить, что он написал, простыми словами и по шагам. Два эффекта. Во-первых, вы поймёте код. Во-вторых, объясняя, помощник нередко сам находит несостыковку и исправляется.

Второй приём — попросить его самого назвать слабые места: где этот код сломается и каких проверок в нём не хватает. Отвечает он на такое честно.

Сколько времени закладывать

Если написание заняло полчаса, на проверку закладывайте столько же. Ощущение «сэкономил час» возникает ровно до первого разбирательства, почему в отчёте не сходятся цифры.

Экономия при этом остаётся реальной: в эксперименте GitHub на 95 разработчиках задача с помощником закрывалась за 1 час 11 минут вместо 2 часов 41 минуты — и это уже с учётом того, что результат доводили до рабочего состояния.

Глава 9. Где вайбкодинг не работает

Коротко

Вайбкодинг помогает не везде. Есть задачи, где он экономит половину времени, и задачи, где он не экономит ничего, а риск добавляет.

По оценке McKinsey, на понятных и небольших задачах генерации нового кода помощник экономит около половины времени разработчика, а на сложных и непривычных выигрыш падает до 10% и меньше. Эта разница и определяет границу.

Где не работает

- Редкие и специфичные для вашей компании вещи. Помощник силён на том, что встречалось часто. Ваш самописный обмен, доставшийся от подрядчика семь лет назад, ему не встречался ни разу.
- Большие изменения в старой системе. Чтобы переделать то, что связано со всем остальным, надо держать в голове всю систему. Помощник видит фрагмент.
- Задачи, где важна каждая цифра. Расчёт зарплаты, налоги, начисления, медицинские дозировки. Не потому что помощник обязательно ошибётся, а потому что цена ошибки несопоставима с экономией часа.
- Производительность под нагрузкой. Помощник пишет код, который работает. Код, который работает на миллионе записей и не кладёт базу, — другая задача, и она требует понимания, как устроены данные именно у вас.
- Безопасность. Помощник не знает вашей модели угроз. Он напишет вход по логину и паролю, но не подумает, кто и что должен видеть после входа.
- Архитектура. Решение, как устроить систему, чтобы её можно было развивать три года, помощнику отдать нельзя. Он ответит уверенно, но это будет усреднённый ответ, а не ответ про ваш случай.

Где выигрыш максимальный

Для равновесия — обратная сторона.

- Типовые обработки данных: собрать, отфильтровать, посчитать, выгрузить.
- Новый небольшой кусок, не связанный с остальным.
- Разбор чужого кода: объяснить, что здесь происходит.
- Перевод с одного языка на другой при переезде.
- Черновик документации и комментариев к готовому коду.
- Мелкие правки по понятному описанию ошибки.

Честно о том, чего вайбкодинг не отменяет

Он не отменяет систему контроля версий, тестирование, код-ревью и приёмку. Наоборот: чем быстрее появляется код, тем нужнее становится всё, что его проверяет.

Он не отменяет знание предметной области. Помощник не знает, что в вашей компании скидка не может быть больше 30%, а заказ без склада не оформляется. Эти правила приносит человек.

Он не отменяет ответственности. За код, который ушёл в работу, отвечает тот, кто его туда отправил.

Как решить в конкретном случае

Один вопрос, который заменяет все остальные: если этот код тихо ошибётся, я об этом узнаю — и во что это обойдётся?

Узнаю сразу и обойдётся дёшево — отдавайте помощнику. Узнаю через месяц из отчёта или от клиента — пишите руками или проверяйте так, как проверяют критичный код.