



**SERVICE
LAB**

Общество с ограниченной
ответственностью «КваСервисЛаб»

info@qaservicelab.ru qaservicelab.ru +79207910079
300000, г. Тула, ул. Тургеневская, д. 69, литер А1, офис 522

5 ошибок тестирования ИТ-продуктов, в которых бизнес боится себе признаться

Практический гайд для ИТ-директоров, технических
руководителей и СТО, которые устали терять деньги из-за багов,
но не знают, с чего начать изменения



Зачем этот гайд

[Мы написали](#) этот материал для всех, кто управляет разработкой в стартапах, продуктовых ИТ-компаниях, внутренних ИТ-подразделениях крупных производственных холдингов.

Нас объединяет общая боль — пропасть между ожидаемым и реальным качеством продукта, которую очень сложно устранить. Часто причина не в технологиях, а в сложных вопросах, которые не принято обсуждать на рабочих встречах.

В этом гайде вы узнаете:

- 5 системных ошибок в организации тестирования (QA, Quality Assurance — обеспечение качества), которые обычно маскируются под «экономия» и «привычные процессы»;
- как эти ошибки проявляются на практике;
- как они влияют на бизнес: финансовые потери, ущерб репутации и трата времени.

Содержание

[Ошибка №1. Экономия на независимом тестировании](#)

[Ошибка №2. Недостаток экспертизы в команде](#)

[Ошибка №3. Недостаток QA-ресурсов](#)

[Ошибка №4. Отсутствие метрик и документации](#)

[Ошибка №5. Устаревшие процессы тестирования](#)

[Бонусная ошибка. Замена QA-экспертизы ИИ-инструментами](#)

[Почему с этими ошибками сложно справиться без внешней команды](#)

[Проверьте себя: 5 признаков, что вам нужна внешняя QA-экспертиза](#)

Ошибка №1. Экономия на независимом тестировании

Как выглядит. В компании нет выделенного QA-инженера. Тестированием занимаются другие сотрудники по остаточному принципу или сами разработчики. Второй вариант встречается особенно часто.

Почему бизнес боится себе в этом признаться. Кажется, что так дешевле: не нужно нанимать отдельного специалиста. Плюс есть удобный аргумент: «Мы пишем юнит-тесты, они все проверяют».

Реальные последствия. Разработчик не может объективно протестировать свой код из-за эффекта «замыленного глаза». Он проверяет сценарии так, как задумывал, а не так, как их будет использовать реальный пользователь или злоумышленник.

Юнит-тесты проверяют отдельные блоки кода, но не охватывают:

- пользовательские сценарии end-to-end;
- пограничные случаи в интеграциях;
- поведение системы под нагрузкой;
- UX-логику и регрессию после обновлений.

По данным отраслевых исследований, значительная доля критических багов обнаруживается уже после релиза — на продакшене, а не на этапе тестирования.

Последствия одного инцидента:

- Дополнительные часы работы команды на исправление ошибок.
- Простой сервиса.
- Штрафы по SLA, если они есть в договоре.
- Ущерб репутации.

Как исправить. Разделите ответственность: разработчик пишет код, а тестировщик проверяет результат. Независимая проверка — это другой взгляд, а не недоверие.

Разработчики должны писать юнит-тесты, но итоговую приемку должен проводить отдельный человек.

Если нет возможности содержать штатную QA-команду, можно [привлекать](#) внешних специалистов точечно: на релизы, регрессию, приемку. Это дешевле, чем постоянный штат, и эффективнее, чем тестировать силами разработчиков.

Ошибка №2. Недостаток экспертизы в команде

Как выглядит. В компании есть QA-отдел, причем немаленький — из 4–5 человек. Они стараются, много работают и закрывают задачи, но качество продукта при этом не улучшается, критические ошибки все равно утекают на продакшен, а сроки тестирования не сокращаются. Руководителя не устраивает результат работы команды, но он не решается на замены.

Почему бизнес боится себе в этом признаться. Признать низкую эффективность команды — значит поставить под сомнение решения ее руководителя или собственное решение о найме. К тому же увольнять людей неприятно, а нанимать новых долго, дорого и энергетически затратно. Проще оставить все как есть и надеяться, что ребята подрастут.

Реальные последствия. Слабая команда создает иллюзию контроля качества. Руководство уверено, что продукт протестирован. На деле проверены очевидные сценарии, а для исследования нестандартных случаев QA-специалистам не хватает опыта. Это приводит к багам на проде, которые «непонятно как пропустили».

Часто тимлид или старший разработчик берет на себя часть работы QA, что приводит к неэффективному использованию времени дорогих специалистов.

Как исправить. Оцените эффективность команды, ответив на вопросы:

- Сколько ошибок выявляется до и после релиза?
- Сколько времени тимлид или разработчики тратят на проверку работы тестировщиков?
- Насколько покрыты нетривиальные сценарии в тест-кейсах?

Если ответы неутешительные, оптимизируйте команду. Исходя из нашей практики, два опытных тестировщика уровня Middle+ с развитыми навыками автоматизации и тест-дизайна способны покрывать объем задач, с которым не справляются четыре менее опытных специалиста.

На переходный период используйте внешнюю QA-команду. Это надежный мост между текущим состоянием и тем, куда вы движетесь.

Главное: не стоит бояться этого решения из-за кажущейся жесткости. Оставить слабых сотрудников — не доброта, а риск, который может вылиться в финансовые и репутационные потери.

Ошибка №3. Недостаток QA-ресурсов

Как это выглядит. Предположим, вы провели оптимизацию команды тестирования и качество пошло вверх. Но со временем возникла новая проблема: проект вырос, над ним работают уже 15–20 разработчиков, а проверяют по-прежнему 2–3 QA-инженера.

Это сильные специалисты. Они отлично знают продукт, быстро тестируют, находят нетривиальные ошибки. Но перед релизами QA-команда в аврале, слегка затягивает спринты и не успевает полностью прогнать регресс. На ретро другие сотрудники говорят, что тестирование — узкое горлышко. На следующем ретро то же самое.

Почему бизнес боится себе в этом признаться. Признать, что команде не хватает рук — значит косвенно обвинить хороших людей в том, что они не вывозят. А они вывозят. Просто объем работы больше, чем физически могут охватить текущие сотрудники. К тому же расширение команды кажется долгим и дорогим процессом.

Реальные последствия. Когда сильных QA-инженеров мало, происходит неизбежное: из-за недостатка времени тестирование становится выборочным.

На практике это означает:

- Регрессия проводится частично, могут возникать баги в стабильных модулях.
- Новые функции проверяются в ущерб регрессии.
- Специалисты работают на износ, что приводит к выгоранию и уходу.
- QA тормозит разработку и задерживает релизы.

Как исправить. Оцените соотношение QA/Dev. Количество QA-инженеров зависит от сложности проекта, числа поддерживаемых платформ и навыков специалистов. Кроме того, чем более гибкая разработка и частые релизы, тем [больше QA-инженеров](#) нужно проекту.

Ориентир для команд с преимущественно ручным тестированием и без автоматизации: [1 QA на 3–4 разработчика](#). Если у вас соотношение 1 к 6 и выше, следует пересмотреть нагрузку QA-команды.

В проектах с развитой автоматизацией и зрелым CI/CD соотношение может быть другим, но даже там стоит проверить, не стало ли тестирование узким горлышком.

Дальше выберите способ восполнения дефицита: найм QA в штат, аутстаф, аутсорс.

Варианты не равнозначны:

Вариант	Скорость подключения	Стоимость	Когда подходит	Особенности
Найм в штат	6–10 недель: поиск + найм + онбординг	Высокая: ФОТ HR / кадрового агентства + ФОТ сотрудника + налоги и взносы	Постоянная высокая нагрузка, долгосрочный продукт	Полное погружение в продукт и высокая управляемость. Финансовые обязательства постоянны (даже при снижении нагрузки). Масштабирование команды требует нового цикла найма
Аутстаф	2–4 недели: подбор специалиста + онбординг на проекте	Средняя: ежемесячная оплата = ставка в час × норма рабочих часов в месяце. Итоговая стоимость включает ФОТ сотрудника + затраты на менеджмент	Нужен специалист под вашим управлением без найма в штат	Оплачивается все рабочее время по производственному календарю, независимо от загрузки задачами. Риск простоев — на заказчике. При долгосрочном привлечении ставка в час ниже, чем при разовых заказах
Аутсорс	1–2 недели: подбор специалистов + онбординг на проекте	Средняя: оплата по фактическим трудозатратам (Time & Material) либо по предварительной оценке (Fixed price)	Пиковая нагрузка, конкретный проект, срочный релиз	Более быстрое подключение к проекту за счет распределения задач между сотрудниками подрядчика. Онбординг проводится однократно, затем команда подрядчика делится знаниями самостоятельно

Ошибка №4. Отсутствие метрик и документации

Как это выглядит. QA-инженеры проверяют функционал по интуиции и памяти, тест-кейсы устарели или отсутствуют, отчеты по тестированию ограничены короткими комментариями «посмотрел, все ок».

Руководство не понимает:

- Что проверили перед релизом.
- Что осталось непроверенным.
- Сколько времени занимает тестирование.
- Как менялось качество продукта от версии к версии.

Почему бизнес боится себе в этом признаться. Выстраивание процессов — работа, которая «не производит результата здесь и сейчас». В условиях горящих дедлайнов документация и

метрики воспринимаются как лишняя бюрократия. Кажется, лучше потратить время на проверку фич, чем на написание тест-кейсов.

Реальные последствия. Без метрик и документации тестировщики работают «вслепую» — не видят проблем, не могут доказать эффективность и системно улучшать качество. На практике это означает:

- Невозможно оценить риски релиза.
- Сложно ускорить тестирование.
- Регрессия становится формальностью.
- Нет данных для принятия решений.

Как исправить. Начните с малого: пишите тест-план перед каждым релизом и краткий отчет после. Это занимает 2–3 часа, но дает другой уровень понимания происходящего.

Дальше внедрите систему управления тестированием и начните собирать базовые метрики. Если нет ресурса выстроить систему с нуля, [привлеките](#) внешнюю команду с релевантным опытом.

Базовые метрики, которые должны быть в любой QA-команде:

Метрика	Зачем нужна
Defect Escape Rate (доля ошибок, пропущенных на продакшен)	Одна из ключевых метрик качества тестирования. Показывает, сколько проблем не обнаружили до релиза. Высокий процент говорит о слабых процессах, низкий — признак надежного покрытия.
Дефекты по приоритетам (критические/высокие / средние / низкие)	Дает понимание приоритетов: какие блоки нужно исправлять срочно, какие по остаточному принципу. Помогает управлять рисками и общаться с разработкой.
Время тестирования по типам (ручное / автоматическое / регрессия)	Позволяет оптимизировать ресурсы: где ускориться за счет автоматизации, где уменьшить регресс, как распределять нагрузку между ручным и автоматическим тестированием.
Requirements Coverage (покрытие требований тест-кейсами)	Показывает, насколько полно протестирован функционал. Применима после того, как выстроена базовая тестовая документация. Без этой метрики есть риск пропустить критические требования и не понять, что еще нужно проверять.

Ошибка №5. Устаревшие процессы тестирования

Как это выглядит. Разработка перешла на Agile и микросервисы, внедрила CI/CD, но QA-процессы остались прежними: ручное тестирование, регрессия раз в спринт, автоматизация

только в планах. Разработчики деплоят часто, тестировщики не успевают проверять релизы. Разрыв между скоростью Dev и скоростью QA растет.

Почему бизнес боится себе в этом признаться. Переработка QA-процессов — системное изменение, которое затрагивает всю команду. Это переговоры, сопротивление, временное снижение скорости. Гораздо проще оставить все как есть и надеяться, что «как-нибудь справимся».

Реальные последствия. Устаревшие QA-процессы в современной команде — это не просто неэффективность. Это системный риск. Конкретные симптомы отставания QA:

- Тестирование стало узким горлышком: разработка готова, а QA еще проверяет предыдущий релиз.
- При двухнедельном спринте регрессия занимает более 3 дней (признак того, что процесс не масштабируется вместе с командой).
- Автоматизированные тесты отсутствуют или давно не поддерживаются.
- QA не участвует в планировании и проектировании и узнает о задаче, когда разработка уже завершена.

Что должно быть в современном QA-процессе:

- Shift Left. Тестировщики подключаются на ранних этапах разработки продукта. Такой подход существенно снижает стоимость исправления ошибок: устранение бага на этапе требований обходится в 5–100 раз дешевле, чем после релиза в продакшен.
- Автоматизация регресса. Критические сценарии покрываются автотестами, которые запускаются при каждом деплое. Это сокращает время регрессии с дней до часов.
- Интеграция QA в CI/CD. Тесты запускаются автоматически в пайплайне, критические падения блокируют деплой в продакшен.
- Нагрузочное тестирование — не разовое событие, а регулярная практика перед значимыми релизами.

Как исправить. Начните с аудита текущих процессов: где QA встраивается в цикл разработки, что автоматизировано, каков Defect Escape Rate. Это даст понимание, насколько велик разрыв между текущим состоянием и тем, что должно быть.

Дальше пошаговая модернизация: сначала автоматизировать smoke-тесты, потом регрессию по критическим модулям, затем интегрировать в пайплайн.

Если внутри нет компетенций для такой трансформации — это тот случай, когда **внешняя экспертиза дает максимальный ROI (Return on Investment — окупаемость инвестиций)**: процессы выстраиваются быстрее по проверенным паттернам.

Кейс из нашей практики. У заказчика из сферы ИБ вырос объем разработки, релизы стали выходить чаще и QA-команда не успевала качественно тестировать новый функционал. Мы за 6 месяцев перестроили QA-процессы компании.

По нашей оценке, количество ошибок разной степени критичности, выходящих на продакшен, снизилось на 70%. Параллельно сократилось время проверки фич и число авралов перед релизами. Это положительно сказалось и на рабочей атмосфере, и на продуктивности команды.

Бонусная ошибка. Замена QA-экспертизы ИИ-инструментами

Как это выглядит. Компания внедряет инструменты для тестирования на базе искусственного интеллекта (ИИ, AI) и начинает сокращать QA-команду или урезать QA-бюджет. Число созданных ИИ артефактов растет, но качество продукта остается прежним или ухудшается.

Почему бизнес боится себе в этом признаться. Тезис «мы используем AI» звучит прогрессивно на любом совещании. Признать, что ИИ не решил проблему качества, значит объяснить, почему деньги потрачены, а ошибки на проде никуда не делись. Кроме того, сокращение QA под предлогом ИИ-автоматизации — быстрая экономия, которая легко обосновывается.

Реальные последствия. ИИ-инструменты правда полезны для тестирования ПО. Они способны усилить специалиста, но **не заменят того, кто хорошо умеет тестировать или знает продукт.**

ИИ генерирует тест-кейсы на основе предоставленных данных: требований, документации, кода. Если сценарий не задокументирован, он не попадет в тест-кейсы. Отчет покажет высокое покрытие, пайплайн останется зеленым, и ни у кого не возникнет ощущения, что что-то упущено. Проблема обнаружится позже, когда непроверенный сценарий воспроизведет реальный пользователь и у него не пройдет платеж или не сработает нужная функция.

Опытный QA-инженер понимает продукт и может предвидеть неожиданные ситуации, поэтому спросит: «Что произойдет, если...?», даже если такого сценария нет в требованиях.

Как исправить. Используйте искусственный интеллект как инструмент усиления команды, а не замену:

- ИИ генерирует черновики тест-кейсов → опытный QA оценивает их полноту, добавляет нестандартные сценарии и расставляет приоритеты с учетом реальных рисков продукта.
- ИИ автоматизирует рутинные проверки → QA-инженер фокусируется на сложных сценариях, граничных случаях и архитектурных рисках, которые ИИ не видит.
- ИИ ускоряет регрессию → QA определяет, что критично проверять и каким результатам можно доверять.

Важно понимать: ИИ-инструменты требуют сопровождения. Автотест, написанный с помощью AI, не будет обновляться сам при изменениях в продукте. Без человека, который следит за актуальностью тестовой базы, автотесты деградируют: начинают падать по случайным причинам или, что хуже, перестают ловить реальные баги.

Еще один принципиальный момент: **ИИ хорошо работает только там, где уже выстроены зрелые процессы** и есть четкие требования, структурированная документация, понятные критерии приемки. Если этого нет, ИИ масштабирует хаос, а не устраняет его. Сначала процессы — потом инструменты. Если процессы еще не выстроены, внешняя команда поможет создать ту самую основу, на которой ИИ-инструменты начнут давать реальный результат, а не красивые отчеты.

Почему с этими ошибками сложно справиться без внешней команды

Чаще всего проблемы остаются нерешенными не из-за отсутствия желания, а по следующим причинам:

- Внутренняя команда не замечает собственных «слепых пятен».
- Нет ресурса на перестройку процессов — все сотрудники заняты текущими задачами.
- Не хватает узкой экспертизы (нагрузочное тестирование, автоматизация, специфическая сфера).

Аутсорсинговая модель в этом контексте перестает быть просто «услугой» и становится стратегическим рычагом: вы платите за снижение числа ошибок на продакшене, прозрачную метрику покрытия и скорость проверки гипотез.

Проверьте себя: 5 признаков, что вам нужна внешняя QA-экспертиза

- ☐ Разработчики сами тестируют свой код перед релизом.
- ☐ Качество работы внутренней QA-команды вас не устраивает, хотя людей в ней достаточно.
- ☐ Перед каждым релизом QA-инженеры в аврале и не успевают прогнать регресс полностью.
- ☐ В проекте нет измеримых метрик качества и актуальной тестовой документации.
- ☐ QA-процессы не пересматривались больше года, несмотря на рост продукта и команды.

Если отметили хотя бы один пункт, вам будет полезен взгляд со стороны. **Приходите к нам на бесплатную диагностику QA-процессов.**

Наши инженеры:

- Познакомятся с вашей архитектурой и процессами.
- Зададут точечные вопросы, которые помогут выявить реальные риски.
- Дадут четкие рекомендации для роста эффективности ваших QA-процессов.

Мы в QA Service Lab занимаемся вопросами обеспечения качества ИТ-продуктов с 2019 года. Работаем с ИТ-компаниями, системными интеграторами и корпоративными ИТ-подразделениями по всей России. В нашем штате инженеры уровня Middle и выше с опытом в сложных сферах бизнеса (от ИБ до нефтедобычи), которые говорят с вашей командой разработки на одном языке.

[Записаться на диагностику→](#)