

Лекция 4.

**Инструменты для организации
тестирования**

Почему важна организация тестирования?

- Тестирование без структуры ведёт к хаосу
- Риски: упущенные дефекты, дублирование, неясные результаты
- Организация повышает эффективность, покрытие и воспроизводимость

Основные инструменты организации тестирования

- Чек-листы
- Тест-кейсы
- Наборы (сююты) тест-кейсов
- Системы управления тестированием

Чек-листы

Что такое чек-лист

- Структурированный список проверок без детализации шагов
- Пример: «Проверить вход по email и паролю»
- Используется на ранних этапах, для smoke-тестов, exploratory-тестирования

- **Плюсы:** быстро создаются, гибкие, легко обновляются
- **Минусы:** зависят от опыта, плохо воспроизводимы, сложно передать

Формы чек-листов

Неупорядоченный список

🚩 Когда порядок пунктов НЕ важен

- Проверка полноты набора значений, опций, состояний
- Подходит для статических списков

📄 Пример: Допустимые статусы заказа

- Создан
- Оплачен
- В обработке
- Отправлен
- Доставлен
- Отменён

✅ Особенности:

- Можно проверять в любом порядке
- Главное — убедиться, что **все значения присутствуют и корректно отображаются**
- Часто используется при тестировании выпадающих списков, фильтров, справочников

Упорядоченный список

Когда последовательность шагов ВАЖНА

- Отражает логику пользовательского сценария
- Каждый шаг зависит от результата предыдущего

Пример: Быстрая регистрация через email

1. Перейти на страницу регистрации
2. Ввести email и нажать «Продолжить»
3. Получить письмо с кодом подтверждения
4. Ввести код на сайте
5. Установить пароль
6. Попасть в личный кабинет

Особенности:

- Нарушение порядка = невозможность выполнить сценарий
- Часто служит основой для **низкоуровневых тест-кейсов**
- Используется в smoke-тестах и ключевых user journey

Многоуровневый список

📌 Когда нужно отразить структуру функциональности

- Группировка по модулям, подсистемам, сценариям
- Улучшает навигацию и планирование покрытия

📄 Пример: Авторизация

- **Успешный вход**
 - По email + пароль
 - Через Google
 - Через Apple ID
- **Ошибки входа**
 - Неверный пароль
 - Несуществующий email
 - Пустые поля
- **Восстановление доступа**
 - По email
 - По SMS
 - Ограничение частоты запросов

✅ Особенности:

- Позволяет видеть **полноту покрытия** на уровне разделов
- Легко превращается в **набор тест-кейсов**
- Идеален для анализа требований и планирования тестирования

Хорошие и плохие чек-листы

Свойства хорошего чек-листа


- **Логичность** — соответствие реальным сценариям
- **Последовательность и структурированность** — группировка по темам
- **Полнота** — включает позитивные, негативные и граничные случаи
- **Неизбыточность** — без дублирования
- **Важно:** чек-лист — это основа (требования) для разработки тест-кейсов

Пример хорошего чек-листа

Контекст: Функционал «Вход в систему»

Авторизация

- Успешный вход с валидным email и паролем
- Ошибка при неверном пароле
- Ошибка при несуществующем email
- Блокировка после 3 неудачных попыток
- Сброс пароля по ссылке из email
- Вход с email в разных регистрах ([user@example.com ↗](#) / [USER@EXAMPLE.COM ↗](#))
- Ограничение длины поля email (254 символа — по RFC)

 Этот чек-лист можно напрямую использовать для создания **низкоуровневых тест-кейсов**.

Пример хорошего чек-листа

Авторизация

- Успешный вход с валидным email и паролем
- Ошибка при неверном пароле
- Ошибка при несуществующем email
- Блокировка после 3 неудачных попыток
- Сброс пароля по ссылке из email
- Вход с email в разных регистрах ([user@example.com ↗](#) / [USER@EXAMPLE.COM ↗](#))
- Ограничение длины поля email (254 символа — по RFC)

✔ Соответствует всем свойствам хорошего чек-листа:

- **Логичность:** пункты отражают реальные действия пользователя
- **Последовательность и структурированность:** сгруппированы по смыслу
- **Полнота:** включены позитивные, негативные и граничные сценарии
- **Неизбыточность:** нет повторяющихся проверок

Пример плохого чек-листа

Контекст: Функционал «Вход в систему»

- Проверить логин
- Попробовать зайти
- Всё должно работать
- Проверить, что нельзя зайти без пароля
- Проверить вход
- Убедиться, что система не ломается
- Попробовать разные комбинации
- Проверить логин ещё раз

! Такой чек-лист **не помогает спланировать покрытие**, не подходит для передачи другому тестировщику и **не даёт основы для тест-кейсов**.

Контекст: Функционал «Вход в систему»

- Проверить логин
- Попробовать зайти
- Всё должно работать
- Проверить, что нельзя зайти без пароля
- Проверить вход
- Убедиться, что система не ломается
- Попробовать разные комбинации
- Проверить логин ещё раз

- **Логичность** — соответствие реальным сценариям
- **Последовательность и структурированность** — группировка по темам
- **Полнота** — включает позитивные, негативные и граничные случаи
- **Неизбыточность** — без дублирования

Пример плохого чек-листа

- Проверить логин
- Попробовать зайти
- Всё должно работать
- Проверить, что нельзя зайти без пароля
- Проверить вход
- Убедиться, что система не ломается
- Попробовать разные комбинации
- Проверить логин ещё раз

× Нарушены ключевые свойства:

- **Нелогичность:** абстрактные, расплывчатые формулировки
- **Отсутствие структуры:** пункты хаотичны
- **Неполнота:** нет негативных/граничных сценариев
- **Избыточность:** дублирование и размытые повторы

Плохой и хороший чек-лист

- Проверить логин
- Попробовать зайти
- Всё должно работать
- Проверить, что нельзя зайти без пароля
- Проверить вход
- Убедиться, что система не ломается
- Попробовать разные комбинации
- Проверить логин ещё раз

Авторизация

- Успешный вход с валидным email и паролем
- Ошибка при неверном пароле
- Ошибка при несуществующем email
- Блокировка после 3 неудачных попыток
- Сброс пароля по ссылке из email
- Вход с email в разных регистрах ([user@example.com ↗](#) / [USER@EXAMPLE.COM ↗](#))
- Ограничение длины поля email (254 символа — по RFC)

Тест-кейсы

Что такое тест-кейс?

- Набор входных данных, условий, шагов и ожидаемого результата для проверки конкретного поведения ПО
 - Цель: воспроизводимость, проверка соответствия требованиям, поиск дефектов
 - Всегда фокусируется на одной логической гипотезе
-
- **Высокоуровневый:** общий сценарий без конкретных данных
 - **Низкоуровневый:** детализированный, с данными и явным ожидаемым результатом
 - Высокоуровневые кейсы — основа для планирования; низкоуровневые — для выполнения

Для чего нужны тест-кейсы

Тест-кейсы — не формальность, а **ключевой инструмент управления качеством** в проекте. Их основные цели:

✓ Структуризация и систематизация тестирования

- Без чётко описанных проверок крупный проект быстро погружается в хаос:
 - одни и те же сценарии проверяются многократно,
 - другие — упускаются полностью.

✓ Измерение и повышение тестового покрытия

- Тест-кейсы — основной источник данных для метрик:
 - покрытие требований,
 - покрытие функциональности,
 - покрытие пользовательских сценариев.
- Без них метрики «покрытия» теряют смысл.

Для чего нужны тест-кейсы

✓ **Контроль выполнения плана**

- Сколько тест-кейсов запланировано?
- Сколько уже написано?
- Сколько выполнено/провалено?
- Это основа для прогнозирования и принятия решений.

✓ **Уточнение требований и выявление неоднозначностей**

- Процесс написания тест-кейсов — это форма **анализа требований**.
- Часто именно при создании тест-кейса обнаруживается, что требование неполное, противоречивое или нереализуемое.

✓ **Улучшение взаимопонимания между командами**

- Тест-кейсы — «живые» примеры поведения системы.
- Для заказчика или разработчика они часто **понятнее сухих требований**.

Для чего нужны тест-кейсы

✓ Долгосрочное хранение знаний

- Информация не теряется при уходе сотрудников.
- Новые члены команды быстрее вникают в логику продукта.

✓ Возможность регрессионного и повторного тестирования

- Без фиксированных тест-кейсов невозможно **гарантировать**, что исправление одного бага не сломало другую функцию.

✓ Быстрое введение новых сотрудников в проект

- Достаточно дать доступ к тестовой документации — и новичок может начать работать уже в первый день.

Вывод: Тест-кейсы — не «бюрократия», а **инвестиция в стабильность, прозрачность и масштабируемость** проекта.

Структура тест-кейса

- ID
- Название / описание
- Предусловия
- Шаги
- Входные данные
- Ожидаемый результат
- Постусловия (опционально)

Структура тест-кейса

UG_U1.12	A	R97	Галерея	Панель загрузки	Загрузка картинки (имя со спецсимволами) Приготовление: создать непустой файл с именем #\$_%^&.jpg. 1. Выбрать вкладку «Загрузить». 2. Нажать кнопку «Выбрать». 3. Выбрать из списка подготовленный файл. 4. Нажать кнопку «ОК». 5. Нажать кнопку «Добавить в галерею».	<ol style="list-style-type: none">1. Вкладка «Загрузить» становится активной.2. Появляется диалоговое окно браузера выбора файла для загрузки.3. Имя выбранного файла появляется в поле «Файл».4. Диалоговое окно файла закрывается, в поле «Файл» появляется полное имя файла.5. Выбранный файл появляется в списке файлов галереи.
----------	---	-----	---------	-----------------	--	--

Идентификатор

Приоритет

Связанное с тест-кейсом требование

Заглавие (суть) тест-кейса

Ожидаемый результат по каждому шагу тест-кейса

Модуль и подмодуль приложения

Исходные данные, необходимые для выполнения тест-кейса

Шаги тест-кейса

Набор тест-кейсов (test suite)

- Группа тест-кейсов, объединённых по:
 - функциональному модулю
 - типу тестирования (smoke, регрессия)
 - приоритету (High/Medium/Low)

Зачем нужны наборы тест-кейсов

- Упрощают планирование и запуск
- Обеспечивают логическое структурирование покрытия
- Повышают управляемость тест-процесса

Качественный тест-кейс

- Атомарность
- Однозначность
- Воспроизводимость
- Актуальность
- Независимость
- Лаконичность
- Правильный уровень детализации
- Учёт негативных и граничных сценариев

Определение:

Тест-кейс должен проверять **ровно одну логическую гипотезу** или одно требование.

Почему важно?

- Если кейс проверяет несколько функций, при падении непонятно, **что именно сломалось**.
- Упрощает анализ результатов и локализацию дефектов.

Пример плохого (неатомарного) кейса:

«Войти в систему, создать заказ и оплатить его»

→ Здесь три проверки: авторизация, создание заказа, оплата.

Пример хорошего (атомарного) кейса:

«Проверить, что пользователь может создать заказ после успешного входа»

→ Одна цель: создание заказа (при условии, что вход уже выполнен — это предусловие).

Однозначность

Определение:

Любой тестировщик должен понимать **одинаково**, что делать и чего ожидать.

Почему важно?

- Исключает субъективность и интерпретацию.
- Обеспечивает согласованность при работе в команде.

Пример неоднозначного кейса:

«Проверить, что всё работает быстро»

→ Что значит «быстро»? 1 секунда? 5 секунд?

Пример однозначного кейса:

«После нажатия кнопки “Сохранить” форма должна закрыться, а запись — появиться в списке **в течение 2 секунд**»

→ Чёткий, измеримый критерий.

Определение:

Тест-кейс должен давать **один и тот же результат** при каждом запуске в одинаковых условиях.

Почему важно?

- Позволяет подтвердить наличие/устранение дефекта.
- Критически важно для регрессионного тестирования.

Пример невоспроизводимого кейса:

«Иногда при входе появляется ошибка»
→ Нет чётких условий, данных или шагов.

Пример воспроизводимого кейса:

— Предусловие: пользователь не авторизован
— Шаги: ввести email **user@test.com** ↗, пароль **12345**, нажать «Войти»
— Ожидаемый результат: появляется сообщение **«Неверный пароль»**
→ Любой может повторить — и получить тот же результат.

Определение:

Тест-кейс должен соответствовать **текущей версии требований и функциональности**.

Почему важно?

- Устаревшие кейсы ведут к ложным срабатываниям или пропуску реальных багов.
- Загрязняют отчётность и снижают доверие к тестированию.

Пример неактуального кейса:

Требование изменилось: теперь пароль должен быть **не менее 8 символов**,
но кейс всё ещё проверяет минимальную длину **6 символов**.

Рекомендация:

- Связывайте тест-кейсы с ID требований (например, REQ-101).
- Обновляйте кейсы при каждом изменении функциональности.

Определение:

Тест-кейс должен **успешно выполняться сам по себе**, без зависимости от других кейсов.

Почему важно?

- Позволяет запускать кейсы в любом порядке.
- Упрощает изоляцию проблем и параллельное тестирование.

Пример зависимого кейса:

ТС_02: «Удалить созданный в ТС_01 заказ»

→ Не работает, если ТС_01 не был выполнен или упал.

Пример независимого кейса:

ТС_02:

— Предусловие: создан тестовый заказ с ID=TEST-123

— Шаг: удалить заказ TEST-123

— Ожидаемый результат: заказ исчез из списка

→ Всё необходимое — в предусловии (или создаётся в шагах).

Пример плохого тест-кейса

- Расплывчатое название
- Отсутствие входных данных
- Неизмеримый ожидаемый результат («всё должно работать»)

Типичные ошибки в тест-кейсах

- Слишком общие формулировки
- Отсутствие приоритизации
- Дублирование пунктов
- Нет чётких критериев прохождения
- Игнорирование негативных сценариев

Слишком общие формулировки + отсутствие критериев прохождения

Пример плохого тест-кейса:

- **Название:** Проверить поиск
- **Шаги:**
 1. Перейти на страницу поиска
 2. Ввести запрос
 3. Нажать «Найти»
- **Ожидаемый результат:** «Результаты должны отображаться корректно»

Что не так? Как исправить?

Слишком общие формулировки + отсутствие критериев прохождения

Пример плохого тест-кейса:

- **Название:** Проверить поиск
- **Шаги:**
 1. Перейти на страницу поиска
 2. Ввести запрос
 3. Нажать «Найти»
- **Ожидаемый результат:** «Результаты должны отображаться корректно»

× Ошибки:

- Что такое «корректно»? Нет измеримого критерия.
- Какой запрос вводить? Нет входных данных.
- Что считается успехом?

Как исправить:

Указать конкретный запрос (например, «ноутбук»), ожидаемое количество результатов (≥ 1), сортировку, соответствие фильтрам и время отклика (< 2 с).

Дублирование + отсутствие приоритизации

Примеры (из одного проекта):

- **TC_SEARCH_01:** Проверить, что поиск работает
- **TC_UI_15:** Убедиться, что поиск показывает результаты
- **TC_REG_07:** Проверить функцию поиска при регрессии

Что не так? Как исправить?

Дублирование + отсутствие приоритизации

Примеры (из одного проекта):

- **TC_SEARCH_01:** Проверить, что поиск работает
- **TC_UI_15:** Убедиться, что поиск показывает результаты
- **TC_REG_07:** Проверить функцию поиска при регрессии

× Ошибки:

- Все три кейса проверяют одно и то же — **дублирование**.
- Непонятно, какой из них **приоритетный** (High/Medium/Low).
- При падении всех трёх создаётся иллюзия множества багов.

Как исправить:

Объединить в один чёткий тест-кейс с ID **TC_SEARCH_01**, присвоить приоритет **High**, и использовать его в нужных наборах (smoke, регрессия и т.д.), а не дублировать.

Игнорирование негативных сценариев

Пример тест-кейса из реального проекта:

- **Название:** Успешный поиск товара
- **Шаги:** Ввести «телефон», нажать «Найти»
- **Ожидаемый результат:** Отображаются товары с названием «телефон»

Что не так? Как исправить?

Игнорирование негативных сценариев


Пример тест-кейса из реального проекта:

- **Название:** Успешный поиск товара
- **Шаги:** Ввести «телефон», нажать «Найти»
- **Ожидаемый результат:** Отображаются товары с названием «телефон»

 Позитивный сценарий — **есть**.

✗ Но полностью отсутствуют:

- Поиск по несуществующему запросу («абракадабра» → «Ничего не найдено»)
- Пустой запрос
- Спецсимволы, SQL-инъекции, очень длинный запрос
- Поиск по пробелам или цифрам

 **Риск:** система падает или показывает ошибку 500 при нестандартном вводе — но тестировщик этого **не проверил**, потому что негатива в кейсах нет.

 **Рекомендация:**

На каждый позитивный сценарий — минимум 1–2 негативных. Включайте их в чек-листы и тест-кейсы **обязательно**.

Типичные ошибки в тест-кейсах

- Несколько проверок в одном кейсе
- Отсутствие предусловий
- Шаги без данных
- Ожидаемый результат неявный
- Использование жаргона

Несколько проверок в одном кейсе + отсутствие предусловий

Пример плохого тест-кейса:

- **Название:** Проверить заказ
- **Шаги:**
 1. Зайти на сайт
 2. Войти в аккаунт
 3. Добавить товар в корзину
 4. Оформить заказ
 5. Оплатить заказ
 6. Проверить email с подтверждением
- **Ожидаемый результат:** Всё работает

Что не так? Как исправить?

Несколько проверок в одном кейсе + отсутствие предусловий

Пример плохого тест-кейса:

- **Название:** Проверить заказ
- **Шаги:**
 1. Зайти на сайт
 2. Войти в аккаунт
 3. Добавить товар в корзину
 4. Оформить заказ
 5. Оплатить заказ
 6. Проверить email с подтверждением
- **Ожидаемый результат:** Всё работает

× Ошибки:

- **Несколько проверок:** авторизация, корзина, оформление, оплата, email → **5 функций в одном кейсе!**
- **Нет предусловий:** неясно, есть ли у пользователя аккаунт, есть ли товар в наличии
- **Неявный результат:** «всё работает» — не измеримо

 **Последствие:** При падении — неизвестно, **что именно сломалось**. Невозможно локализовать дефект.

 **Решение:** Разбить на отдельные атомарные кейсы с чёткими предусловиями.

Шаги без данных + неявный ожидаемый результат

Пример:

- **Название:** Проверить регистрацию
- **Предусловия:** —
- **Шаги:**
 1. Перейти на страницу регистрации
 2. Заполнить поля
 3. Нажать «Зарегистрироваться»
- **Ожидаемый результат:** Пользователь зарегистрирован

Что не так? Как исправить?

Шаги без данных + неявный ожидаемый результат

Пример:

- **Название:** Проверить регистрацию
- **Предусловия:** —
- **Шаги:**
 1. Перейти на страницу регистрации
 2. Заполнить поля
 3. Нажать «Зарегистрироваться»
- **Ожидаемый результат:** Пользователь зарегистрирован

× Ошибки:

- **Какие поля? Какие данные?** Нет входных значений (email, пароль, имя)
- **Какой формат email?** Допустимы ли спецсимволы?
- **Что значит «зарегистрирован»?** Попадает на главную? Получает email? Виден в админке?

Как должно быть:

— Email: test_user@domain.com ↗

— Пароль: StrongPass123!

— Ожидаемый результат: появляется уведомление «Аккаунт создан», пользователь перенаправлен на личный кабинет, в базе появляется запись с этим email.

Использование жаргона + отсутствие контекста

Пример из реального проекта:

- **Название:** Пробросить юзера в личный кабинет после логина через SSO
- **Шаги:**
 1. Залогиниться через SAML
 2. Убедиться, что юзер попал в ЛК без редирект-петли
- **Ожидаемый результат:** Все ок, редиректов нет

Что не так? Как исправить?

Использование жаргона + отсутствие контекста

Пример из реального проекта:

- **Название:** Пробросить юзера в личный кабинет после логина через SSO
- **Шаги:**
 1. Залогиниться через SAML
 2. Убедиться, что юзер попал в ЛК без редирект-петли
- **Ожидаемый результат:** Все ок, редиректов нет

× Ошибки:

- **Жаргон:** «юзер», «ЛК», «все ок», «пробросить», «редирект-петля»
- **Неясно новому тестировщику** или коллеге из другой команды
- **Нет технического контекста:** что такое SAML? Как выглядит «петля»?

Как исправить:

Использовать **формальный язык**:

— «После аутентификации через единый вход (SAML IdP) пользователь должен быть перенаправлен в личный кабинет **один раз**, без повторных HTTP-редиректов (проверить в DevTools → вкладка Network)».

 **Правило:** Тест-кейс должен быть понятен **любому тестировщику**, даже без знания внутреннего сленга команды.



Типичные ошибки в тест-кейсах

- Смешение целей (например, функциональность + регрессия)
- Отсутствие логической группировки
- Избыточность
- Нет управления версиями
- Игнорирование приоритетов

Типичные ошибки в тест-кейсах

Смешение целей + отсутствие логической группировки + игнорирование приоритетов

📄 Пример: Набор «Тестирование_всего_v1»

(реальный фрагмент из плохо организованного проекта)

- TC_LOGIN_01 — Успешный вход (High)
- TC_SEARCH_05 — Поиск по тегам (Medium)
- TC_REG_02 — Регистрация нового пользователя (High)
- TC_UI_10 — Цвет кнопки «Отмена» (Low)
- TC_PAY_03 — Оплата картой (High)
- TC_HELP_01 — Ссылка на FAQ работает (Low)
- TC_LOGIN_02 — Ошибка при неверном пароле (High)

Что не так? Как исправить?

Типичные ошибки в тест-кейсах

✘ Ошибки:

- Смешение целей: smoke, функциональность, UI, документация — всё в одном
- Нет логической группировки: авторизация перемешана с оплатой и справкой
- Игнорирование приоритетов: Low и High запускаются одинаково, нет фильтрации

⚠ Последствие:

- Невозможно быстро запустить **smoke-тест** перед релизом
- Тестировщик тратит время на низкоприоритетные проверки в критической ситуации

✅ Как должно быть:

Отдельные наборы:

- **Smoke_Auth** (только High: вход/регистрация)
 - **Regression_Payment**
 - **UI_Checks** (Low-priority, запускается раз в неделю)
-

Типичные ошибки в тест-кейсах

Избыточность + отсутствие управления версиями

📄 Ситуация:

Команда обновила требования: теперь подтверждение email при регистрации отменено.

Но в системе остались следующие наборы:

- **Набор А:** содержит TC_REG_01 — «Проверить подтверждение email»
- **Набор В:** содержит копию того же кейса под ID TC_EMAIL_CONFIRM
- **Набор С:** содержит устаревшую версию TC_REG_01 (без пометки об изменении)

Что не так? Как исправить?

Типичные ошибки в тест-кейсах

Избыточность + отсутствие управления версиями

✘ Ошибки:

- **Избыточность:** один и тот же сценарий дублируется под разными ID
- **Нет управления версиями:** неясно, какая версия актуальна, когда кейс был изменён
- **Нет связи с требованиями:** невозможно отследить, что требование REQ-REG-04 устарело

⚠ Последствие:

- Тестировщики продолжают проверять **удалённую функцию**
- Метрики покрытия искажены
- Возникают ложные баг-репорты

✅ Решение:

- Хранить кейсы в системе с **историей изменений** (TestRail, Xray и др.)
- Присваивать **уникальные ID** и связывать с требованиями
- Удалять или **архивировать** устаревшие кейсы с комментарием: «*Удалено в v2.1, требование отменено*»

Как избежать ошибок в тест-кейсах

- Проводить ревью тестовой документации
- Использовать шаблоны и стандарты
- Обучать команду
- Интегрировать с системой требований
- Регулярно актуализировать тесты

Системы управления тестированием

Определение:

Система управления тестированием (Test Management System, TMS) — это специализированное ПО для:

- Хранения и структурирования тест-кейсов
- Формирования наборов (test suites)
- Планирования и выполнения тестовых прогонов (test runs)
- Отслеживания результатов и метрик
- Интеграции с баг-трекерами и CI/CD

Основные цели TMS:

- Устранить хаос в Excel-файлах
- Обеспечить совместную работу команды
- Автоматизировать отчётность
- Связать тесты с требованиями и дефектами

 Без TMS масштабирование тестирования в крупных проектах становится крайне затруднительным.

✓ Управление тестовой документацией

- Создание, редактирование, версионирование тест-кейсов
- Поддержка предусловий, шагов, ожидаемых результатов

✓ Группировка и организация

- Наборы (suites), подпроекты, теги, приоритеты
- Иерархическая структура (папки → модули → кейсы)

✓ Планирование и выполнение

- Test cycles / test runs
- Назначение тестировщиков
- Фиксация результатов: Pass / Fail / Blocked


✓ Интеграции

- Jira, Bugzilla, Azure DevOps (для багов)
- Jenkins, GitLab CI (для автоматизации)
- Confluence (для требований)

✓ Аналитика и отчётность

- Покрытие требований
- Прогресс выполнения
- Тренды по качеству (regression pass rate и др.)

Популярные TMS

Система	Особенности	Подходит для 
TestRail	Гибкость, мощная аналитика, отличная интеграция с Jira	Средних и крупных команд
Xray (Jira)	Встроен в Jira, поддержка BDD, Gherkin, автоматизации	Команд, уже использующих Atlassian
Zephyr Scale / Squad	Zephyr Scale — для автоматизации; Squad — для ручного тестирования	Enterprise-проектов
qTest	Часть Tricentis, поддержка end-to-end тестирования	Команд, работающих с автоматизацией
Kiwi TCMS	Open-source, простой интерфейс	Небольших команд, стартапов, учебных проектов

 Выбор зависит от: размера команды, бюджета, уровня зрелости процессов, наличия автоматизации.

Сравнение TMS с электронными таблицами

Критерий	Excel / Sheets	TMS 
Версионирование	✗ Только вручную (файлы v1, v2...)	✓ Автоматическое, с историей изменений
Совместная работа	✗ Конфликты при редактировании	✓ Одновременная работа, блокировки
Связь с багами	✗ Только ссылки	✓ Прямая интеграция (1 клик → баг в Jira)
Метрики	✗ Ручной подсчёт	✓ Автоматические дашборды
Повторное использование	✗ Копирование ячеек	✓ Переиспользование кейсов в разных наборах
Безопасность	✗ Файл может быть удалён	✓ Резервное копирование, права доступа

 **Вывод:** Excel подходит для старта, но TMS — обязательна при переходе к зрелым процессам.

Практические рекомендации

✓ 1. Начинайте с чек-листа — завершайте тест-кейсами

— Используйте чек-листы на этапе анализа требований для быстрого охвата функциональности.

— На их основе создавайте структурированные, атомарные тест-кейсы по мере стабилизации требований.

✓ 2. Соблюдайте принцип «один кейс — одна проверка»

— Это упрощает локализацию дефектов и повышает воспроизводимость.

— Избегайте «эпических» кейсов вроде «Протестировать весь заказ».

✓ 3. Всегда формулируйте ожидаемый результат явно и измеримо

— ✗ «Система должна работать корректно»

— ✓ «После нажатия “Оплатить” отображается страница с надписью “Заказ принят”, статус в ЛК меняется на “Оплачен”»

✓ 4. Не забывайте про негативные и граничные сценарии

— На каждый позитивный кейс — минимум 1–2 негативных:

- Пустые поля
- Спецсимволы
- Максимальная/минимальная длина
- Неверные форматы данных

✓ 5. Используйте ID, теги и приоритеты

— Пример ID: `TC_AUTH_01` (модуль + номер)

— Теги: `smoke`, `regression`, `security`

— Приоритеты: `High` / `Medium` / `Low` — для гибкого планирования запусков

✓ 6. Связывайте тесты с требованиями

— Каждый тест-кейс должен ссылаться на ID требования (например, `REQ-LOGIN-03`).

— Это позволяет отслеживать покрытие и быстро реагировать на изменения.

✓ 7. Регулярно актуализируйте тестовую документацию

— Удаляйте или помечайте как «устаревшие» кейсы, относящиеся к удалённой функциональности.

— Проводите ревью после каждого крупного релиза.

✓ 8. Используйте TMS, а не Excel, на проектах дольше 2–3 недель

— Даже бесплатные системы (Kiwi TCMS, TestLodge) дают огромное преимущество в управлении и отчётности.

💡 Главный принцип:

Тестовая документация — это не отчёт для менеджера, а рабочий инструмент команды.

Она должна быть понятной, актуальной и полезной — иначе её перестанут использовать.

Чек-лист для проверки тест-кейса

◆ 1. Структура и ясность

- Есть уникальный и понятный ID (например, `TC_AUTH_01`)
- Название **отражает суть проверки** («Успешный вход с валидным email и паролем»)
- Все шаги **пронумерованы** и записаны в повелительном наклонении («Нажать», «Ввести», «Проверить»)
- Используется **формальный язык**, без жаргона и сленга

◆ 2. Атомарность и фокус



- Тест-кейс проверяет **ровно одну гипотезу** (нет нескольких целей в одном кейсе)
- Не дублирует другие тест-кейсы в наборе

Чек-лист для проверки тест-кейса

◆ 3. Предусловия и данные

- Указаны **предусловия** (например, «Пользователь зарегистрирован», «Товар в корзине»)
- Все **входные данные конкретны**:
 - Email: `user@test.com`
 - Пароль: `Pass123!`
 - Количество: `999` (для граничных значений)

◆ 4. Ожидаемый результат

- Результат **явный, измеримый и однозначный**
 -  «Система работает нормально»
 -  «Отображается сообщение “Неверный пароль”, поле подсвечено красным»
- Учтены **визуальные, функциональные и технические аспекты** (UI, API, БД — если применимо)

Чек-лист для проверки тест-кейса

◆ 5. Полнота сценариев

- Для позитивного сценария есть соответствующие негативные/граничные кейсы в наборе
 - Пустой ввод
 - Спецсимволы / SQL-инъекции
 - Превышение лимитов (макс. длина, количество попыток и т.д.)

◆ 6. Управляемость

- Присвоен приоритет (High / Medium / Low)
- Добавлены теги (например, smoke , regression , security)
- Указано требование, которое покрывается (например, REQ-LOGIN-02)

Чек-лист для проверки тест-кейса

◆ 7. Актуальность

- Кейс соответствует текущей версии спецификации
- Не относится к удалённой или изменённой функциональности

◆ 8. Независимость

- Может быть выполнен без запуска других тест-кейсов
- Все необходимые данные создаются в предусловиях или шагах

Эффективное тестирование начинается с организации.

✓ **Чек-листы** — ваш первый инструмент анализа:

- Они помогают быстро охватить функциональность,
- Выявить пробелы в требованиях,
- И стать основой для полноценных тест-кейсов.

✓ **Тест-кейсы** — основа воспроизводимого и управляемого тестирования:

- Качественный кейс — атомарный, однозначный, независимый,
- Он позволяет измерять покрытие, отслеживать прогресс и уверенно проводить регрессию.

✓ **Наборы тест-кейсов** — ключ к масштабированию:

- Правильно сгруппированные, приоритизированные и актуальные наборы
- Позволяют запускать нужные проверки в нужное время — от smoke-теста до полного регресса.

✓ **Системы управления тестированием (TMS)** — не роскошь, а необходимость:

- Они превращают хаотичные Excel-файлы в живую, управляемую экосистему качества.

Главный вывод:

Хорошо организованное тестирование — это системный подход, а не случайные проверки.

Инвестиции в структуру, документацию и инструменты многократно окупаются:

- снижением количества упущенных дефектов,
- повышением скорости реакции на изменения,
- и устойчивостью команды даже при росте сложности проекта.

Помните:

Лучший тест-кейс — тот, который понятен коллеге, актуален сегодня и помогает найти баг завтра.