Тестирование приложений: описание и чек-лист

В этой статье изложен опыт компании Infoshell по тестированию приложений. Речь пойдёт о тестировании в спринте и в проектной работе, предрелизном тестировании и других вопросах.

Тестирование в спринте

В первый день спринта (выделенного на одну функцию или часть продукта периода) необходимо создать тест-кейсы и автотесты. Или отредактировать их, если текущий спринт не первый в цепочке.

Текст-кейс и автотест

Тест-кейс — это документация тестировщика. Это список действий, который нужно совершить, чтобы достичь поставленной цели. Он включает в себя, в среднем, 5 пунктов:

- уникальный ID, который необходим для оптимизации поиска и хранения кейса;

- название документа — предмет тестирования или основная тема теста. Раздел включает также краткое описание сути работ. Не пишите большие названия и всегда делайте их однотипными, так вам будет проще найти документ, если вы потеряете или забудете его ID;
- предусловия, или условия, которые не относятся к функционалу на проверке прямо, но обязательны для выполнения. Например, заполнение профиля доступно только зарегистрированному, авторизованному и подтверждённому пользователю. В таком случае в предусловиях тест-кейса «Заполнить профиль» должно быть: «пользователь зарегистрирован», «пользователь авторизован», «пользователь подтверждён»;
- список шагов — список действий, которые позволят достичь поставленной цели: протестировать разработку; результат — краткое описание наиболее вероятного, по мнению тестировщика, результата после совершения всех шагов тестирования. В заключении может быть три варианта: «положительный», «отрицательный» и «тест блокирован».

Положительный — фактический результат равен ожидаемому. Отрицательный — если не равен и была найдена какая-то ошибка. «Тест блокирован» — невозможность выполнения шагов. Это означает ошибку, которую необходимо найти и указать в результате.

По желанию в текст-кейс можно добавлять другие пункты. Например, историю редактирования, в которой нужно указывать, кем и когда этот кейс был изменён, что было убрано или добавлено.

При этом в тест-кейсе не должно быть нечётких формулировок, лишних деталей и описаний, умалчиваний или неточностей в описании шагов и результата. Ещё одно важное условие — каждый кейс должен быть независим от остальных. Держите это в голове, так как тест-кейсы и автотесты пишутся на каждую функцию, и начать связывать их автоматически очень легко.

Автотест — код, который пишет разработчик для проверки работоспособности приложения. Позволяет не только сэкономить время на поиск и исправления багов, но и увидеть конкретную строчку кода с ошибкой.

Тест-кейсы и автотесты нужны для того, чтобы проверить, как продукт будет вести себя при разных действиях пользователя: ожидаемых и неожиданных, логичных и нелогичных, правильных и неправильных.

> **Также полезно**
>
> Гид по профессии тестировщик: чем занимается специалист в сфере QA, сколько зарабатывает, что надо знать и где учиться.

Тестирование функциональности

В большинстве случаев это тестирование проводится вручную. Этапа три, они идут по порядку:

1. Дымовое тестирование. Цель — проверить базовую функциональность приложения. Продумайте поведение пользователя, затем начните тестировать приложение. Сделайте позитивный тест и выполните целевое действие. Пусть это будет подписка на email-рассылку.
2. После него начните тест негативный: попытайтесь «обмануть» программу, кликайте на разные кнопки, иконки, изображения и отслеживайте, как она реагирует на ваши действия. Продукт не должен открывать никаких лишних окон, картинок. Он также не должен давать никакой информации при нажатии «в пустоту».
3. Если всё прошло хорошо, переходите к следующему этапу. Если хотя бы в одном из этих тестов вы нашли ошибку, приложение нужно отправить на доработку с описанием обнаруженных багов в баг-репорте. Назначьте задачу на исправление и ждите устранения ошибки, после чего повторите дымовое тестирование.

Регрессионное тестирование

Необходимо для того, чтобы проверить, исправили ли разработчики найденные баги. Ещё одна цель регрессионного тестирования — отслеживание того, как внесённые изменения повлияли на работу других

частей приложения и его поведение в целом.

Сэм Канер, профессор программной инженерии Технологического института, директор Образовательного и исследовательского центра тестирования программного обеспечения Флориды, выделяет три основных типа этого теста:

- регрессия багов. Цель тестировщика — увидеть и доказать, что найденная на более ранних этапах ошибка не была исправлена в полной мере или в принципе;
- регрессия старых багов. Цель тестировщика — увидеть и доказать: разработчики изменили код или данные так, что старые баги начали снова воспроизводиться;
- регрессия побочного эффекта. Цель тестировщика — увидеть и доказать: разработчики изменили код или данные так, что нетронутые части приложения сломались из-за этого (или приложение в целом вышло из строя).

Полное тестирование по тест-кейсам. На третьем этапе тестировщик проверяет все функции, которые описаны в его тест-кейсах. Когда результат по каждому из них будет положительным, тестирование можно считать оконченным.

Тестирование в проектной работе

В проектной работе применяют преимущественно регрессионное тестирование. Это обусловлено тем, что тест в данном случае проводят на заключительных этапах. Предположим, запланировано четыре спринта. Тогда тестировщик включается после третьего.

Особое внимание при тестах в проектной работе стоит уделить тестированию взаимодействия. Это функциональный тест, который проверяет способность продукта взаимодействовать с компонентами или системами: от одного и больше. Оно включает в себя тестирование совместимости и интеграционное тестирование.

Тестирование совместимости — нефункциональный тест, цель которого — проверить, корректно ли работает приложение в определённом окружении. Это может быть аппаратная платформа, различные ОС, браузеры и расширения.

Интеграционное тестирование — фаза теста ПО, где отдельные модули программы объединяют и тестируют в группе. Интеграционное тестирование можно автоматизировать с помощью систем непрерывной интеграции (например, Jenkins, TeamCity, Travis CI, Gitlab CI, Circle CI, GoCD или другие).

Предрелизное тестирование

Это особенный, наиболее важный спринт и тестирование. Перед релизом продукт необходимо «прогнать» ещё раз, чтобы убедиться в отсутствии багов (по крайней мере, больших) наверняка.

Сначала нужно провести регрессионное тестирование. В идеале процесс разработки должен быть построен так, чтобы для теста перед релизом оставались только мелкие функции, баги которых не требуют много времени для устранения. Также важно учитывать, чтобы возможные исправления не могли повлиять на другие части продукта и его поведение в принципе. Ведь исправлять всё перед релизом попросту некогда.

Если процесс был спланирован правильно, регрессионное тестирование ограничится проверкой случайных изменений в коде с прошлого спринта. Если нет — случиться может всё, что угодно. Но вероятность того, что вы спокойно завершите работу над приложением в срок, крайне мала.

После регрессионного начинайте тестирование внедрённых багфиксов (исправленных ошибок). Сейчас тестировщик должен проверить, есть ли какие-то негативные последствия от исправления багов, найденных с помощью регрессионного теста, или нет. А также были ли эти ошибки вообще исправлены разработчиками.

Затем идёт тестирование интеграции патча (код, который добавили разработчики для устранения ошибок). Фактически, это тест результата двух предыдущих этапов. Тестировщик пытается понять, не вредит ли патч

приложению, и насколько хорошо он «встал» в систему. Помимо патчей на данном этапе проверяют все дополнения, которые были внесены в проект за последнее время. Для этого используются юнит-тесты.

Юнит-тест — автотест для небольшой части кода, которая отвечает за конкретную функцию приложения. Юнит-тесты подают значения. Тест считается пройденным, если программа обрабатывает их верно — так, как было задумано тестировщиком. Если реакция приложения не совпадает с запланированной, тест считается не пройденным. Но разработчики понимают, в какой части кода находится ошибка, и исправляют её.

Это не вся польза юнит-тестов. Они могут очень помочь в борьбе с багами после обновлений. Ведь возвращение к старому коду требует много времени на разбор и сравнение его с новым, а юнит-тест покажет область проблемы сразу.

Последний этап — это финальное тестирование продукта. Оно включает в себя проверку всех функций приложения с учётом спецификации или бэклога, которую команда согласовала с заказчиком.

Тестирование на поддержке

Это тестирования, которые происходят после релиза продукта. Они нужны только в том случае, когда заказчик попросит добавить новый функционал или договорится с компанией о дальнейшем обслуживании приложения и исправлении новых багов.

Во время тестирования на поддержке нужно проверять сборку: основной функционал после обновлений. Когда тестировщик обнаружит баг, он должен описать его. Дальнейшая работа над проблемой будет происходить по принципу тестирования во время спринта.

Вывод и чек-лист

Важно придерживаться единообразия стандартов, так как это залог стабильной работы отдела тестировщиков. Мы используем описанные выше методики и принципы, чтобы оптимизировать все процессы, экономить время и силы сотрудников, упрощать разработку и не позволять багам проникать в пост-релиз.

Для удобства можно использовать в работе чек-лист.

- Этап 1: тестирование в спринте.
  - тест-кейс и автотест — создаётся документация тестировщика и автотесты;
  - тестирование функциональности — ручной тест (позитивный и негативный);
  - регрессионное тестирование — отслеживаются исправления;
- Этап 2: тестирование в проектной работе — регрессионное тестирование (проверка совместимости, интеграционный тест).
- Этап 3: предрелизное тестирование — регрессионные тесты, проверка багфиксов, интеграции патчей, юнит-тесты.
- Этап 4: тестирование на поддержке — выполняется при обновлении функциональности.