«Проект 1: регрессионное тестирование»

Проект 1

1. Проект в TMS: https://tms.devexplab.ru/projects/lukashenkovaleryia-14-qa-plus-sprint1/testruns/2
2. Баг-репорты в YouTrack: https://lukashenkovaleryia.youtrack.cloud/projects/be858708-db4a-46c6-90d3-c0a4bea6b9ab
3. Выводы о работе:

   На тестирование ушло в общей сложности неделя с перерывами. Если брать в расчет непрерывное время работы, то часов 20 с несколькими переделками.
   Тестовое окружение: Chrome (версия 114.0.5735.133) on Windows 10 Home (версия 22H2, сборка 19045.3086)
   В процессе тестирования удалось найти 9 багов:
   - 1 критический
   - 2 серьезных
   - 5 средних
   - 1 незначительный

   Из них три бага (1 критический и 2 серьезных) фактически не дают пользователю взаимодействовать с приложением.
   На мой взгляд, в описания некоторых тест-кейсов можно внести изменения в названии, так как не полностью указано “Где?” и “Когда” происходит действие тест-кейса:

№	Текущее название	Предлагаемое название
3	Обновление полей "Имя" и "Занятие"	Обновление полей "Имя" и "Занятие" **в окне редактирования профиля**
13	Отображается текст из блока пользователя в поле "Имя"	**В окне редактирования профиля** в поле "Имя" отображается текст с главной страницы
12	Отображается текст из блока пользователя в поле "Занятие"	**В окне редактирования профиля** в поле "Занятие" отображается текст с главной страницы
8	При пустом поле "Занятие" кнопка "Сохранить" не активна	При пустом поле "Занятие" кнопка "Сохранить" **в окне редактирования**

		профиля не активна

Тестируемое приложение небольшое, но в более крупных было бы логично создать тест-сьюты с указанием необходимого модуля (окно редактирования профиля, фотоальбом и т.д.) и тогда краткие заголовки были бы уместны. У нас в тест-ране мы тестируем, по сути, два модуля: окно редактирования профиля и главную страницу, и это вызывает некоторое недопонимание. Разработчик не должен догадываться, где тестировщик тестировщик нашел ошибку, а структурированные описания тест-кейсов позволят создавать более четкие баг-репорты разработчикам.

В 15-ом тест кейсе не указано предусловие.

В таком виде приложение выпускать в релиз нельзя, так как оно, по сути не функционирует.

4. Вопросы на рассуждение:

Для разработки приложений есть несколько разных моделей разработки. Выбор модели зависит от специфики проекта, требований и финансовых возможностей заказчика, команды разработки и так далее.
Рассмотрим две из наиболее часто применяемых моделей разработки приложений: из линейных моделей возьмем спиральную, а из гибких SCRUM.

Спиральная модель – это вариант водопадной модели разработки. Она предполагает начальный выпуск продукта с базовой функциональностью MVP (Minimum Viable Product), а затем последовательное добавление в продукт новых функций и исправление появляющихся багов в новых, периодически выпускаемых, версиях продукта. Такую модель разработки используют для больших и дорогостоящих проектов, где требования, стоимость и сроки четко определены. Стадии разработки идут последовательно одна за другой.
Рассмотрим на примере добавления в приложение Яндекс Самокат функции поиска ближайшей станции зарядки и функции поиска по сигнализации.

До начала исполнения циклов разработки, как для линейной модели, определяются:

- команда для разработки;
- сроки и бюджет проекта.

На этапе проектирования команда определяет необходимые требования:

- использование геолокации пользователя для определения его местоположения и более точного поиска зарядной станции;
- использование GPS-трекера в самом самокате для определения его местоположения и поиска на карте ближайшего к пользователю;

- использование звуковых и/или световых эффектов для нахождения самоката;
- добавление нужных кнопок в приложение и связанный с этим дизайн-макет новых/измененных экранов;
- поддержка новых функций на серверах приложения Яндекс Самокат и связанное с этим расширение API для взаимодействия с мобильным приложением

На этапе дизайна прорабатывают требования к новым функциям (как будут выглядеть новые кнопки, где расположены, шаги для их активации и т.д.) и создают детальный дизайн нужных страниц в приложении. Кроме того, программисты продумывают новые функции приложения (как для мобильного приложения, так и для сервера) и определяют с помощью каких компонент в коде, структур данных и алгоритмов будет поддерживаться новая функциональность.
На этапе кодирования программисты пишут код по выработанным требованиям, дизайну и проработанной на предыдущем этапе архитектуре ПО. Пока программисты пишут код, тестировщики готовятся к тестированию:

- изучают требования и дизайн;
- уточняют непонятные моменты;
- проектируют тесты: составляют чек-листы и тест-кейсы.

На этапе тестирования. После окончания разработки кода, тестировщики приступают непосредственно к тестированию. Сначала проведут смоук-тестирование основных функций:

- запускается ли приложение Яндекс Самокат;
- пользователь может войти в свой аккаунт;
- отвечает ли сервер и база данных.

Если на этом этапе, если выявлены блокирующие баги, составляются баг-репорты и приложение отправляется на доработку. После починки багов снова проводим тестирование (ре-тесты).
После смоук-тестирования, если оно прошло успешно, имеет смысл провести санитарное тестирование по отдельным важным компонентам приложения:

- проходит ли оплата по банковской карте;
- происходит ли верификация;
- GPS системы функционируют.

После успешного завершения предыдущих проверок проводим регрессионное тестирование – это полная проверка всего приложения. Для проверки старых функций есть уже готовые тест-кейсы, таким образом, составляют новые тест-кейсы только для новых функций версии программного продукта, проверка которых является приоритетной при регрессионном тестировании.
На основании тестирования составляются баг-репорты, далее чинится код, проводятся ре-тесты. После всех видов тестирования готовят отчет, в котором релиз-менеджер указывает, сколько ошибок удалось найти и исправить. Затем релиз-менеджер на основании полученных результатов принимает решение о выпуске релиза.

Далее рассмотрим гибкую модель: **SCRUM**.

Это методика гибкого управления проектами, которая позволяет структурировать процесс разработки проекта в соответствии с текущими требованиями и возникающими проблемами. В рамках этой модели работа делится на фиксированные отрезки-спринты, которые чаще всего длятся две недели, но не дольше четырех.

Владелец продукта ведет бэклог продукта с общим списком задач на все спринты и детализирует требования к этим задачам вместе с командой.
Бэклог продукта основывается на дорожной карте (стратегии длительного развития продукта) и требованиях к продукту (требования к определенному продукту, в том числе назначение продукта, его возможности, функции и принцип работы).
Затем проводится планирование спринта, на котором владелец продукта ставит цель спринта и команда разработчиков решает, какие работы нужно выполнить в этом спринте для достижения обозначенной цели и как они будут выполняться. На собрании по планированию спринта команда создает бэклог спринта – Sprint Backlog – то есть рабочие задачи на текущий спринт и план их выполнения. Для работы в ходе спринта выбираются задачи из бэклога спринта и меняется их статус с «Открыто» на «В работе» и на «Готово» по мере завершения работ.
В течение спринта команда собирается на ежедневные Scrum- совещания (стендапы), чтобы обсудить ход работы, на которых определяются проблемы, которые могут повлиять на цели спринта. Обычно стендапы длятся не более 15 минут.
После окончания спринта команда показывает выполненную работу (проводит Демо): показывает проект заказчику, владельцу продукта и руководителю спринта (менеджеру или SCRUM-мастеру), вносит коррективы в Бэклог продукта, если требуется.
Ретроспектива спринта. Команда обсуждает результаты спринта, что хорошо работает, а что нет, и какие ошибки нужно исправить. Здесь собирается только команда, без прочих гостей, т.к. принимаемые здесь решения касаются усовершенствования работы самой команды.
Затем можно начинать следующий спринт.
Таким образом, с помощью этого метода разработки можно продукт выпустить быстрее, чем при спиральном методе, когда нужно дожидаться окончания разработки всего продукта в целом.
На примере приложения Яндекс Самокат:

- Разделяем “Поиск ближайшей станции зарядки” и “Поиск самоката по сигнализации” на множество задач и ранжируем их по значимости;
- в ближайший спринт отбираем важнейшие;
- анализируем требования
- уточняем бэклог продукта
- составляем бэклог спринта
- программисты выполняют кодирование, а тестировщики параллельно проектируют тесты
- тестирование (смоук-тестирование - санитарное - регрессионное тестирование - отчет)
- исправление багов

- ре-тест - проверка исправления багов прошлых спринтов
- демо
- релиз
- ретроспектива спринта

По аналогичной схеме планируем спринт по следующим задачам в зависимости от их приоритета.
Реальная работа тестировщиков начнется ближе к середине спринта после окончания основных разработок. Часть некритичных багов может быть перенесена в следующий спринт: пока разрабатывается новое, параллельно чинится старое.

Проект 2: ретест багов

Баги в YouTrack:

https://lukashenkovaleryia.youtrack.cloud/projects/6af4cedb-9756-4730-94f4-62aeb6f2f19d

Выводы о работе:

Баг-репорты были сформулированы недостаточно четко:

- не указано окружение;
- не во всех баг-репортах указано предусловие;
- не прикреплены теги;
- нет скриншотов багов

Было исправлено 9 багов, найдено 2 новых и 3 бага не исправлено. В 3-х предыдущих багрепортах были найдены новые баги, в двух из них баг один и тот же.

Вопросы на рассуждение

Может ли появиться другая ошибка на месте исправленной? Почему?

На месте любой исправленной ошибки может появиться новая по нескольким причинам:

1. разработчику не хватает знаний(новичок);
2. были допущены ошибки при кодировании(опечатки, описки, проблемы копирования/вставки и т.д.);

3. сделали быстрое решение проблемы в ущерб качеству;
4. если ошибка затрагивает архитектуру проекта, то при ее исправлении затрагивается большая область и появляются новые баги;
5. баг-репорт был составлен некорректно или с ошибками и разработчик его неправильно понял.

Если обнаружен новый баг.

Если при ре-тесте обнаружен новый баг, то создается новый баг-репорт, который проходит все циклы с начала:

- open
- in progress
- fixed
- testing
- re-opened
- closed

Что делать тестировщику, если разработчики не исправили несколько найденных багов?

- Проверить написание баг-репорта, могла быть допущена ошибка и вследствие этого разработчик не понял, что исправлять;
- Это не баг, но изменения в документацию не внесены;
- Написать разработчику и уточнить причину
- Если остались открытыми критичные для выпуска версии баги (на взгляд тестировщика), сообщить об этом менеджеру

Может ли случиться такое: разработчик сообщил, что исправил ошибки, но после повторной проверки выяснилось — все баги по-прежнему на месте? Почему?

- Да, если разработчик исправил для новой версии, но не выложил нужные исправления в текущую версию, ре-тесты которой выполняет тестировщик
- Еще может быть, что на стенде тестирования выполнены такие настройки и собраны такие данные, что ошибка здесь все еще воспроизводится, а в среде разработки – нет.