МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА ПО ДИСЦИПЛИНЕ

«ПРОЕКТНАЯ ДЕЯТЕЛЬНОСТЬ»

Лаборатория Тестирования

Куратор проекта: _____ / Семенов Сергей Максимович, Преподаватель/

подпись ФИО, уч. звание и степень

Студент: _____ / Белов Владимир Андреевич , 171-371 /

подпись ФИО, группа

Студент: _____ / Колпаков Алексей Андреевич , 171-372 /

подпись ФИО, группа

Студент: _____ / Камашева Юлия Олеговна , 171-371 /

подпись ФИО, группа

Студент: _____ / Селиванов Сергей Денисович , 171-372 /

подпись ФИО, группа

Студент: _____ /, /

подпись ФИО, группа

Студент: _____ /, /

подпись ФИО, группа

Москва, 2021

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Проект по тестированию мобильного приложения «Юла» от разработчиков «Майл.ру» представляет собой поиск ошибок в работе этого самого приложения, который может помешать пользователю комфортно его использовать.

В данном случае нами была проанализирована система аудио и видео звонков, непосредственно встроенных в приложение и осуществляющихся через него. Команда была распределена на подгруппы, где каждой было выдано своё задание.

Было выявлено 3 направления тестирования: аудио звонки, видеозвонки и звонки внутри приложения при помощи p2p соединения (сеть соединений, где каждый равноправный член сети может хранить и обрабатывать запросы от других участников сети) были проведены работы по выявлению ошибок в их работе.

Для регистрации всех ошибок использовались такие программные средства как:

1. TestReil – инструмент для управления основными процессами тестирования, начиная от регистрации и заканчивая сроками решения и выполнения работы;
2. Google Таблицы – для ведения учёта основных аспектов работы;
3. Trello – программа для менеджмента и учёта сроков выполнения основных работ.

ПЛАН РАБОТ

1 ТЕСТИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Тестирование — процесс, содержащий в себе все активности жизненного цикла, как динамические, так и статические, касающиеся планирования, подготовки и оценки программного продукта и связанных с этим результатов работ с целью определить, что они соответствуют описанным требованиям, показать, что они подходят для заявленных целей и определения дефектов.

Тестировщик — специалист, который занимается тестированием. В его обязанности входит поиск возможных ошибок и сбоев в функционале тестирования объекта, например приложения. Тестировщик моделирует ситуации, вероятные при использовании тестируемого объекта, чтобы потом разработчики могли устранить обнаруженные неполадки.

Дефект(Баг) — отклонение фактического результата от ожиданий наблюдателя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

Тестирование — только один из аспектов обеспечения качества. Тестирование входит в контроль качества, а он входит в процесс обеспечения качества в целом. Чтобы не вдаваться в детали, разделим тестирование и обеспечение качества, поскольку часто в вакансиях указан либо тестировщик, либо инженер по качеству.

Чем занимается тестировщик:

1. Тестирует и анализирует продукт.
2. Составляет тесты для дальнейшего проведения тестирования.
3. Взаимодействует с разработчиками и командой для анализа и исправления дефектов.
4. Составляет отчётную документацию о результатах тестирования на определённых этапах.

Чем занимается инженер по обеспечению качества:

1. Формирует критерии качества и контролирует их соблюдение.
2. Внедряет стандарты.
3. Изучает возможности изменения и улучшения процесса разработки.
4. Обучает новых специалистов.
5. Улучшает коммуникацию в команде.

Другими словами, тестировщик задействован непосредственно в тестировании, а инженер по обеспечению качества включён в выстраивание и улучшение процессов в команде и на проекте в целом.

1.1 Тестовая документация

Тестовая документация включает в себя:

- тест план;
- тестовая стратегия;
- чек-лист;
- тестовый сценарий;
- тестовый комплект;
- пользовательская история (User Story);
- отчет о дефекте.

Тест план (Test Plan) - это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

Тестовая стратегия определяет то, как мы тестируем продукт. Это набор мыслей и идей, которые направляют процесс тестирования.

В стратегии тестирования описывают:

1. Тестовую среду.
2. Анализ рисков проекта.
3. Инструменты, которые будут использовать, чтобы провести автоматизированное тестирование и для других целей.
4. План действий при непредвиденных обстоятельствах.
    - **Пользовательские истории** (User Story) — способ описания требований к разрабатываемой системе, сформулированных как одно или более предложений на повседневном или деловом языке пользователя. Пользовательские истории используются гибкими методологиями разработки программного обеспечения для спецификации требований.

Чек-лист - набор идей по тестированию, разработке, планированию и управлению. А также, это перечень формализованных тестовых случаев в удобном для проведения проверок виде. Тестовые случаи в чек-листе не должны быть зависимыми друг от друга.

Обязательно должен содержать в себе следующую информацию:

- идея проверок;
- набор входных данных;
- ожидаемые результаты;
- булевая отметка о прохождении/непрохождении тестового случая;
- булевая отметка о совпадении/несовпадении фактического и ожидаемого результата по каждой проверке.

Отчёт о дефекте - это документ, описывающий ситуацию или последовательность действий приведшую к некорректной работе объекта тестирования, с указанием причин и ожидаемого результата.

Тест кейсы нашего проекта:

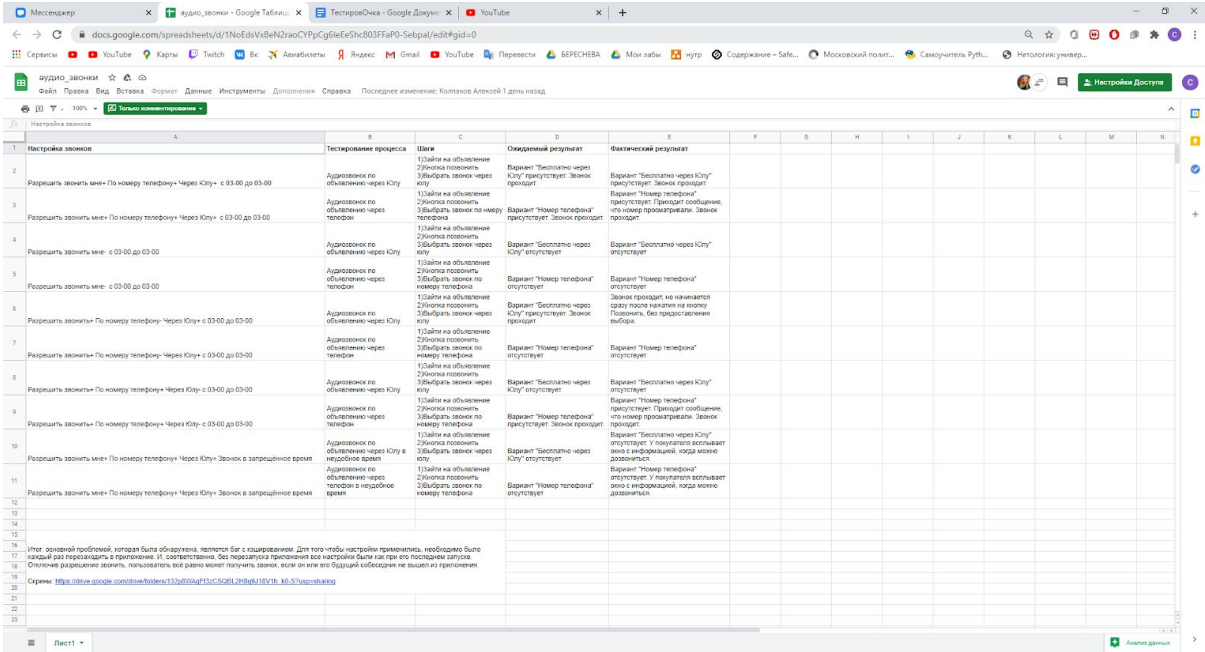


Рисунок - тест кейсы

1.2 Описание дефектов

При создании отчёта о дефекте важно написать название, которое коротко и ёмко отражает суть проблемы.

Хорошо написанное название позволяет разработчику быстрее сориентироваться и устранить дефект. то важно при написании названия отчёта о дефекте:

1. Название должно содержать ответы на следующие вопросы:

- Что? Что происходит или не происходит согласно спецификации или представлению тестировщика о нормальной работе продукта.
- Где? В каком месте интерфейса пользователя или архитектуры программного продукта находится проблема.
- Когда? В какой момент работы программного продукта, по наступлении какого события или при каких условиях проблема проявляется.

2. Содержать предельно краткую, но достаточную для понимания сути проблемы информацию о дефекте.

3. Должно быть достаточно коротким, чтобы полностью помещаться на экране, потому что в системах отслеживания ошибок (bugtracker) конец предложения в поле краткого описания (summary) обрезается или приводит к появлению скроллинга.

4. Должно быть законченным предложением русского или английского (или иного) языка, построенным по соответствующим правилам грамматики.

Например:

● приложение зависает на экране загрузки файла после сохранения текстового файла размером больше 50 Мб;

● данные не сохраняются на форме «Профайл» после нажатия кнопки «Сохранить»;

● поле «Подтвердите пароль» не является обязательным на форме регистрации;

● текст выпадающего списка отображается за пределами выделенной области на главной странице после наведения курсора на меню вкладки «Курсы».

Шаги по воспроизведению также являются ценной информацией в отчёте, поскольку представляют собой руководство к действию для тех, кто будет решать проблему. Шаги должны удовлетворять следующим требованиям:

1. На первом шаге необходимо использовать ссылку на главный домен:

● открыть сайт http://....(Open the site http://...., Go to the site http://...);

2. В шагах нужно отвечать на вопрос «Что необходимо сделать?»:

● нажать кнопку «Найти» (Click the «Найти» button);

● ввести валидный email и пароль (Enter the valid email and the password);

● заполнить необходимые поля валидными данными (Fill the necessary fields with valid data).

3. В шагах необходимо коротко писать, что сделать, куда нажимать, не уточняя названия страницы.

4. Описывается как минимум два шага, но не больше семи-восьми шагов.

5. Если шагов для воспроизведения будет очень много (больше восьми), можно указать предусловия (Preconditions), например: Пользователь авторизован в системе. Товар был добавлен в корзину.

6. В шагах нужно уточнять, на что именно необходимо нажимать (на ссылку, кнопку, логотип).

7. Шаги стоит писать с заглавной буквы.

8. В последнем шаге нужно описать, на какую область посмотреть, на что обратить внимание или что осмотреть:

● осмотреть текст в выпадающем списке (подменю) (Look at the drop-down list (the submenu);

● обратить внимание на «Profile» форму (Take a look at the «Profile» form, Pay attention to the «Profile» form). Фактический и ожидаемый результаты рекомендуется описывать следующим образом:

1. Результаты необходимо описывать информативно (так же, как и тему, по принципу «Что? Где? Когда?», иногда можно использовать «Что?», «Где?»).

2. Сначала описывается фактический результат, затем ожидаемый.

3. Один результат в одном дефекте.

4. Результаты следует описывать полными предложениями, с подлежащим и сказуемым.

5. Результаты необходимо писать с заглавной буквы.

Например:

Фактический результат. Элементы подменю «Курсы» отображаются за границами выпадающего списка в главном меню после наведения курсора на блок «Курсы».

Ожидаемый результат. Элементы подменю «Курсы» находятся в пределах выпадающего списка в главном меню после наведения курсора на блок «Курсы».

Серьёзность (severity) показывает степень ущерба, который наносится проекту существованием дефекта.

Срочность (priority) показывает, как быстро дефект должен быть устранён.

На разных проектах тестировщики могут заполнять только «Серьёзность», а на некоторых и «Срочность». Часто на проектах встречается распределение ответственности при определении срочности и серьёзности дефекта. «Серьёзность» (severity) заполняет тестировщик, а «Срочность» (priority) — руководитель тестирования или проекта, исходя из того, насколько сильно дефект влияет на продукт, какие приоритеты у разработки либо на какой стадии находится разработка проекта. 14 Выделяют следующие градации серьёзности, которые могут различаться в зависимости от проекта:

- Block (блокирующий дефект) — блокирует большую часть работы в проекте;
- Crash (авария) — приводит к падению приложения (или даже операционной системы);
- Major (значительный дефект) — дефект в важном, но не ключевом функционале
- Minor (незначительный дефект) — дефект в дополнительном функционале;
- Tweak (неудобство) — неудобство в использовании, требуется небольшая доработка для повышения удобства использования;
- Text (текст/опечатка) — небольшая текстовая ошибка, опечатка;
- Trivial (тривиальный дефект) — незначительная недоработка. Срочность может быть высокой (High), средней (Medium), низкой (Low). Порядок исправления ошибок по их приоритетам: High → Medium → Low.

Статус дефекта зависит от практики, сложившейся на проекте, и используемой системы отслеживания дефектов. Ориентировочно можно выделить следующие:

● Новый (New) — новый отчёт о дефекте;
● Открыт (Opened) — отчёт о дефекте открыт;
● Назначен (Assigned) — отчёт о дефекте назначен разработчику;
● Исправлен (Fixed) — дефект исправлен;
● Ретест (Retest) — дефект исправлен, требуется проверка тестировщиком;
● Открыт снова (Reopened) — дефект проверен тестировщиком, ошибка не исправлена;
● Закрыт (Closed) — дефект проверен тестировщиком, отчёт о дефекте закрыт.

При создании отчёта о дефектах также важно помнить базовые правила:

1. Старайтесь следовать правилу «Что? Где? Когда?»
2. Одна ошибка — один отчёт о дефекте.
3. Старайтесь быть лаконичными при описании дефекта.
4. Каждый дефект должен быть воспроизведён повторно перед написанием отчёта.
5. Отчёт о дефекте должен быть составлен сразу, не откладывайте на потом.
6. Минимизируйте количество шагов в описании.
7. Пишите техническим языком с применением терминологии, принятой на проекте.
8. Прикрепляйте дополнительные файлы (логи, скриншоты, видео).
9. Старайтесь часть логов прикреплять в описании.

Это поможет при поиске дефектов по сообщениям об ошибках, которое выдаёт приложение.

10.Используя видео, вы можете короче описать дефект и избежать недопонимания.
11.Прикрепляйте ссылки к требованиям, это поможет избежать споров и сэкономить время.
12.Избегайте дубликатов дефектов — прежде чем составлять отчёт о дефекте, спросите у коллег или проверьте в баг-трекере, есть ли уже такой дефект.
13.Указывайте версию ПО и тестовый стенд (окружение), на котором был обнаружен дефект.
14.Попробуйте воспроизвести дефект, следуя вашим собственным шагам.

2 ТЕСТИРОВАНИЕ ВЕБ-СЕРВИСОВ

2.1 Тестирование API

API (сокр. от англ. Application Programming Interface, по-русски обычно говорят «апи», без перевода) —программный интерфейс приложения —описание способов (набор классов, процедур, функций, структур или констант),

которыми одна компьютерная программа может взаимодействовать с другой программой. По сути, API выступает в роли посредника между двумя приложениями или сервисами —предоставляет решения (классы, функции, структуры), реализованные в одном сервисе, и создаёт среду для создания нового приложения с применением этих решений.API может быть разработан для веб-систем, ОС, баз данных и других типов продуктов. Например, программист, который разрабатывает приложения для Android, может использовать Android API для взаимодействия с аппаратной частью. API существует не только у веб-приложений. Он может быть у любого продукта, которому нужно получать данные извне или передавать их.

Целью тестирования API является проверка функциональности, надежности, производительности и безопасности программных интерфейсов. При тестировании API используется программное обеспечение вместо стандартных пользовательских вводов (клавиатуры) и выводов для отправки вызовов API, получения вывода и записи ответа системы. Тесты API сильно отличаются от тестов GUI и не будут фокусироваться на внешнем виде приложения. Основное внимание уделяется уровню бизнес-логики в архитектуре программного обеспечения.

Тестирование API проводят, основываясь на бизнес-логике программного продукта. Тестирование API относится к интеграционному тестированию, а значит в ходе него можно отловить ошибки взаимодействия между модулями системы или между системами. Для тестирования используют специальные инструменты, где можно отправить входные данные в запросе и проверить точность выходных данных.

Подход API тестирования:

- Понимание функциональности программы API и четкое определение объема программы
- Использование методов тестирования, такие как классы эквивалентности, анализ пределов и предположений, и напишите тестовые примеры API
- Записи API должны быть организованы и правильно определены
- Выполнение тестовых примеров и сравнение ожидаемых и фактических результатов.

Инструменты для тестирования API:

- Postman
- Jmeter
- SoapUi
- Katalon Studio

2.2 Тестирование WEB

Веб-приложение – это клиент-серверное приложение, в котором клиентом выступает браузер, а сервером – веб-сервер (в широком смысле). Основная часть приложения, как правило, находится на стороне веб-сервера, который обрабатывает полученные запросы в соответствии с бизнес-логикой продукта и формирует ответ, отправляемый пользователю. На этом этапе в работу включается браузер, именно он преобразовывает полученный ответ от сервера в графический интерфейс, понятный рядовому пользователю.
Основные виды тестирования сайта (веб-приложения)

1. Тестирование функциональности;
2. Тестирование удобства использования;
3. Тестирование интерфейса;
4. Тестирование совместимости;
5. Тестирование производительности и скорости загрузки сайта;
6. Тестирование безопасности.

Остановимся поподробнее на первом пункте
«Клиент-сервер» —это вид архитектуры распределенного приложения. В этой архитектуре есть два типа элементов: клиенты и сервер. Они взаимодействуют определенным способом, который называется протоколом. К одному серверу могут подключаться и одновременно запрашивать нужную информацию миллионы клиентов.
Обычно клиенты и сервер расположены на разных вычислительных машинах и взаимодействуют через сеть посредством сетевых протоколов, но они могут быть и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных или сервисов, а клиенты инициализируют сеансы с серверами. Примеры приложений, использующих модель «клиент-сервер»:
●веб-приложения (сайт Geekbrains.ru, онлайн-магазин Ozon.ru); ●электронная почта
Для взаимодействия с веб приложением необходи проколов http
Протокол HTTP (сокр. от англ. HyperText Transfer Protocol) —протокол прикладного уровня, используемый в веб-приложениях для передачи данных между клиентом и сервером. По умолчанию используется порт 80. HTTP чаще всего использует возможности протокола TCP для пересылки своих сообщений.HTTPS (сокр. от англ. HyperText Transfer Protocol Secure) —расширение протокола HTTP для поддержки шифрования в целях повышения

безопасности. Данные в протоколе HTTPS передаются поверх криптографических протоколов SSL или TLS. HTTPS по умолчанию использует порт 443.HTTP —это так называемый stateless-протокол. Он не имеет состояния, то есть в этом протоколе не существует связи между двумя запросами, которые последовательно выполняются по одному соединению.Для обхода этого ограничения могут применяться Cookie, которые позволяют использовать сессии с сохранением состояния. Cookie добавляются к рабочему потоку благодаря расширяемости заголовков и позволяют сессии на каждом HTTP-запросе делиться состоянием.

Сообщения HTTP —запросы и ответы —имеют одинаковую структуру, которая включает следующие компоненты:1.Стартовая строка, описывающая запрос, который будет реализован, или его статус: успешный или неудачный. Запрос в стартовой строке всегда пишется одной строкой.2.Необязательный набор заголовков HTTP, уточняющих запрос или описывающих тело, включенное в сообщение.3.Пустая строка, которая отделяет метаинформацию от тела запроса.4.Необязательное тело сообщения, содержащее данные запроса (например, содержимое HTML-формы) или данные ответа. Присутствие тела и его размер определяются стартовой строкой и заголовками.

Методы:

- HEAD запрашивает данные с сервера, но в ответе сервера отсутствует тело. Этот метод может использоваться для получения метаданных, проверки наличия ресурса, даты его изменения без необходимости пересылки большого количества данных.
- GET используется для получения запрошенного ресурса.Запросы, использующие метод GET, должны только получать данные и никоим образом не влиять на приложение. Согласно стандарту HTTP, запросы типа GET считаются идемпотентными.
- POST необходим для передачи данных от клиента к серверу. Данные, полученные в теле POST-запроса, принимаются и обрабатываются сервером. Метод POST не считается идемпотентным, то есть многократная отправка одного и того же POST-запроса может создавать новые объекты на сервере (запись в БД, изменение файла и так далее).
- PUT может использоваться для загрузки содержимого из тела запроса на указанный URI. Если по данному адресу существует ресурс, он будет изменён. Если ресурса нет, он будет создан. Разница между PUT и POST заключается в том, что PUT —идемпотентный метод. То есть единичные

и множественные вызовы этого метода с идентичным набором данных будут иметь один и тот же результат выполнения.

- PATCH используется для изменения содержимого по указанному URI. В отличие от PUT, данный метод не может создавать новые ресурсы на сервере.
- DELETE удаляет ресурс по заданному URI.TRACEвозвращает клиенту полученный от него запрос, так что клиент может видеть, какие изменения в запросе были сделаны на промежуточных серверах.
- OPTIONS возвращает список HTTP-методов, которые предоставляет сервер для указанного URI. Этот метод также может использоваться для проверки возможностей всего сервера, если вместо адреса конкретного ресурса указать «*».
- CONNECT запускает двустороннюю связь с запрошенным ресурсом. Метод можно использовать для открытия туннеля. К примеру, метод CONNECT может использоваться для доступа к сайту, который использует SSL (HTTPS).

Код состояния присылается сервером как часть первой строки ответа. Он состоит из трёх цифр, где первая —класс состояния.Всего существует 5 классов:

1. Информационные ответы (100–199).
2. Успешные ответы (200–299).
3. Перенаправление (300–399).
4. Ошибки клиента (400–499).
5. Ошибки сервера (500–599).

2.3 Нагрузочное тестирование

Нагрузочное тестирование - важный элемент комплексного тестирования производительности, цель которого - проанализировать скорость отклика системы на внешний запрос. Этот тест позволяет определить, соответствует ли тестируемое приложение или устройство заявленным требованиям. Кроме того, можно понять, как программа отреагирует, если ее используют несколько пользователей одновременно. Основная идея нагрузочного тестирования - создать определенную нагрузку с помощью программного и аппаратного обеспечения, а затем отслеживать индекс производительности продукта.

Основные этапы нагрузочного тестирования

1. Подготовка — Проводится анализ целей и статистики эксплуатации системы. Определяются бизнес-операции, имеющие значение с точки зрения

нагрузки на систему. Создается и согласуется документ «Методика нагрузочного тестирования», который включает: стратегию тестирования, список и описание тестов, критерии успешного завершения, описание средств мониторинга и инструментов нагрузочного тестирования. Осуществляется подготовка тестовых данных, настраивается мониторинг, наполняется база данных.

2. Проведение — Выполняется запись и отладка скриптов нагрузочного тестирования и реализация сценариев нагрузочного тестирования. С помощью разработанной системы нагрузочного тестирования выполняется тестирование приложений. В рамках нагрузочных испытаний собирается различная статистическая информация, с помощью которой выполняется анализ производительности.

3. Отчет — После проведения нагрузочного тестирования компании клиенту предоставляется отчет, который описывает результаты тестирования, отступления от методики (если имеются), список ошибок, предложения по оптимизации работы системы, общие замечания.

Основными целями нагрузочного тестирования являются:

1. Оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию
2. Оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патч-сетов
3. Оптимизация производительности приложения, включая настройки серверов и оптимизацию кода
4. Подбор соответствующей для данного приложения аппаратной (программной платформы) и конфигурации сервера

Основные инструменты для проведения нагрузочного тестирования:

1. Jmeter
2. LoadRunner
3. Gatling
4. Яндекс.Танк

2.4 Автоматизированное тестирование

Автоматизированное тестирование ПО — процесс тестирования программного обеспечения, при котором основные функции и шаги теста, такие как запуск, инициализация, выполнение, анализ и выдача результата, производятся автоматически с помощью инструментов для автоматизированного тестирования.

Основной целью автоматизации является оптимизация временных и человеческих ресурсов, затрачиваемых на проведение тестирования. Исключен «человеческий фактор». Сильное достоинство. Все мы люди и никто из нас не застрахован от ошибок. Выполняемый же тест-скрипт не пропустит тест по неосторожности и ничего не напутает в результатах.

- Быстрое выполнение – автоматизированному скрипту не нужно сверяться с инструкциями и документациями.
- Меньшие затраты на поддержку – когда скрипты уже написаны, на их поддержку и анализ результатов требуется, как правило, меньшее время чем на проведение того же объема тестирования вручную.
- Отчеты – автоматически рассылаемые и сохраняемые отчеты о результатах тестирования.
- Выполнение без вмешательства – во время выполнения тестов инженер-тестировщик может заниматься другими полезными делами, или тесты могут выполняться в нерабочее время.

В процессе автоматизации можно выделить три основных этапа:

- начальный этап - этап подготовки и планирования. На этом этапе принимается решение о необходимости автоматизированного тестирования, оцениваются потенциальные возможности и экономический эффект, устанавливаются цели и стратегии автоматизации, а также определяются типы тестов, подходящие для автоматизации. На этом же этапе после тщательного изучения свойств, оценки степени соответствия конкретной задаче и оценки ресурсов, необходимых для поддержания нормальной работы, выбирается инструмент автоматизации. В результате первоначальной разработки устанавливаются требования к описанию тестов, проверяется совместимость средств автоматизации и тестируемого программного обеспечения и тестовой среды, а также разрабатываются точные методы оценки затрат на внедрение. этап активной разработки;
- этап поддержки автоматических тестов. Большая часть времени на этом этапе посвящена описанию, разработке, тестированию и выполнению автоматизированных тестов. Уменьшает количество ресурсов, необходимых для разработки общих функций. В случае крупных проектов команда разработчиков тестов может быть значительно увеличена, а успешно завершенная фаза подготовки и планирования гарантирует минимальные риски. При разработке тестов важно обеспечить возможность автоматического документирования обнаруженных ошибок и составления общего отчета по результатам автоматических тестов.

После успешного выполнения автоматических тестов результаты анализируются и при необходимости корректируются. Активная фаза разработки может быть довольно продолжительной, в зависимости от размера проекта.

- этап поддержки автоматических тестов. Для автоматизации тестов выбираются только те тесты, которые тестируют фиксированную часть программы. Однако изменение требований к входу, обновление настроек или структуры среды тестирования может привести к тому, что автоматические тесты будут давать ошибочные результаты. Следовательно, изменения в системе всегда следует отслеживать, и при необходимости автоматизированные тесты следует адаптировать или модифицировать, чтобы поддерживать их в актуальном состоянии.

3 ТЕСТИРОВАНИЕ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Очевидно, что мобильное приложение сильно отличается от настольного. Поэтому необходимо учитывать это при планировании процесса тестирования.

Рассмотрим основные различия между мобильными и настольными приложениями:

- Мобильное устройство – это система, которая не обладает мощными характеристиками. Таким образом, он не может работать как персональный компьютер.
- Тестирование мобильных приложений проводится на мобильных телефонах (Apple, Samsung, Nokia), в то время как настольное приложение тестируется на центральном процессоре.
- У мобильных устройств бывают разные разрешения. Размер экрана мобильного телефона меньше, чем у настольных.
- Выполнение и прием вызовов является основной задачей телефона, поэтому приложение не должно вмешиваться в эту важную функцию.
- Широкий спектр конкретных операционных систем и компонентных конфигураций: Android, iOS, BlackBerry.
- Операционная система мобильного телефона быстро устаревает.

- Мобильные устройства используют сетевые подключения (3G, 4G, Wi-Fi), широкополосное подключение к настольному ПК или Wi-Fi.
- Мобильные устройства постоянно осуществляют поиск сети. Вот почему необходимо протестировать приложение с разной скоростью передачи данных.
- Инструменты, которые хорошо подходят для тестирования настольных приложений, не полностью подходят для тестирования мобильных приложений.
- Мобильные приложения должны поддерживать несколько входных каналов (клавиатура, голос, жесты и т. д.), мультимедийные технологии и другие функции, повышающие их удобство использования.

Тестирование мобильных приложений можно разделить на ручное и автоматизированное. Необходимо понимать, что в краткосрочной перспективе выгоднее использовать ручное, но в долгосрочной - автоматизированное.

Конечно, нет однозначных ответов на то, какую стратегию лучше всего выбрать. Однако сочетание различных вариантов наиболее оптимально. Например, вы можете использовать симуляторы на самых ранних этапах вашего тестирования. Но лучше использовать реальные устройства (физические или облачные) на заключительных этапах. Автоматизированное тестирование предпочтительнее для нагрузочного и регрессионного тестирований.

3.1 Ручное тестирование

Самый большой из недостатков ручного тестирования — человеческий фактор. Он, конечно, влияет на всё, происходящее в команде — и на работу участников, и на события, происходящие на стороне клиента. В случае QA причиной того, что тестировщик будет слабо погружен в процесс и пропустит потенциальную ошибку может стать всё что угодно — недостаток опыта, семейные проблемы или головная боль.

Один и тот же сценарий два тестировщика могут проверить разными способами. Что с этим делать? Важно, чтобы каждый непредусмотренный или отличающийся от ожидаемого результат был зафиксирован в виде кейса. Чтобы любой тестировщик мог проверить его, совершив тот же набор действий. Но и этого может быть мало — если кейс окажется недостаточно подробным, а тестировщик — не разберётся в описании. Гарантировать стопроцентное исключение человеческого фактора, конечно, невозможно — но можно постараться максимально снизить вероятность проблем, которые он вызывает.

Это тоже негативно сказывается на сроках поставки фичи в продакшн и трудозатратах: ведь теперь к периодически проводимым базовым кейсам и регрессии добавляются и “хитрые” кейсы, придуманные тестировщиками в процессе.

Усугубляет ситуацию вероятность того, что часть встреченных багов ещё не будет иметь строгого описания потому что причины их возникновения пока не понятны. Как бороться с такими повторными тестированиями? Либо найти уже источник ошибки и устранить его, либо — всё таки автоматизировать прохождение проблемных кейсов — в этом случае переход к программным тестам будет вполне оправдан как с точки зрения времени, так и финансово. (Нет, это не противоречит вышесказанному — потому что такие ситуации возникают уже в ходе активной разработки и к этому времени вы уже в любом случае развернёте процессы автотестирования).

В любом случае — появление первых кейсов, нуждающихся в регрессивных тестах или релиз второй версии приложения и соответствующее этим событиям масштабирование команды — тот момент, когда автоматизация станет актуальна (но не исключит ручное тестирование новых возможностей). На этом этапе автоматизация уже начнёт экономить время: разработчик сможет сам, ещё до передачи QA-отделу запустить регресс-тесты новой фичи, чтобы убедиться, что она не ломает существующий функционал, а тестировщику не придётся лишний раз проходить по набившим оскомину базовым кейсам. Ещё

одно преимущество внедрения автотестов на этом этапе — их можно запускать по определённому расписанию — каждую ночь, в дни окончания спринтов или при публикации новых сборок приложения.

При этом нельзя забывать несколько вещей:

- создание кейсов и написание автотестов будет требовать времени — закладывайте его в спринты;
- убедитесь, что кейс автотеста прописан хорошо и подробно и описывает возможную проблему или существующий сценарий во всей полноте;
- проверьте, правильно ли работает автотест и действительно ли он проверяет то, что нужно и делает это качественно.

Достоинства ручного тестирования мобильных приложений:

- Это более экономически выгодно в краткосрочной перспективе.
- Ручное тестирование более гибкое.
- Лучшее моделирование действий пользователя.

Недостатки ручного тестирования мобильных приложений:

- Ручные тестовые примеры трудно использовать повторно.
- Менее эффективно выполнение определенной постоянной задачи.
- Процесс тестирования медленный.
- Некоторые типы тестовых случаев не могут быть выполнены вручную (нагрузочное тестирование).

3.2 Автоматизированное тестирование

Автоматизация тестирования помогает решить сразу несколько проблем — в том числе если речь идёт о мобильных приложениях. Вместо того чтобы вручную проводить рутинные трудоемкие процедуры, специалисты могут делегировать значительную их часть фреймворкам. Автоматизация упрощает проверку и помогает ускорить регрессионное тестирование, а также даёт возможность использовать ранее недоступные типы тестирования.

В идеале вы должны настроить как можно больше кейсов, что позволит вам автоматизировать около 80% ваших процессов тестирования. Существуют конкретные тестовые случаи, которые должны быть автоматизированы, список перед вами:

- автоматизируйте самые частотные тест-кейсы;
- автоматизируйте тестовые случаи, которые легко автоматизировать;
- автоматизируйте тест-кейсы, которые имеют предсказуемые результаты;
- автоматизируйте самые утомительные ручные тесты;
- автоматизируйте тест-кейсы, которые невозможно выполнить вручную;
- автоматизируйте тест-кейсы, которые выполняются на нескольких различных аппаратных или программных платформах и конфигурации;
- автоматизируйте часто используемые функции.

Преимущества автоматизированного тестирования приложений:

- Процесс тестирования занимает мало времени.
- Экономичность в долгосрочной перспективе использования.
- Автоматизированные тестовые случаи легко использовать повторно.
- Единственное решение для некоторых видов тестирования (тестирование производительности).
- Результаты испытаний легко доступны.

Недостатки автоматизированного тестирования приложений:

- У некоторых мобильных средств тестирования есть ограничения.
- Процесс тестирования занимает много времени.

Автоматизированное тестирование наименее эффективно в определении удобства пользования, что подводит нас к следующем разделу.

4 ЮЗАБИЛИТИ ТЕСТИРОВАНИЕ

Юзабилити-тестирование – это сценарный план, предложенный специалистом на основе анализа вашего ресурса или виртуального бюро услуг. Цель юзабилити-анализа – экспериментальным путем установить, удобен ли

ресурс для клиентов, а если нет – вывести парадигму требуемых улучшений.

Только на основе юзабилити-тестирование вы сможете понять – какие исходные данные нужно использовать для A/B-тестирования (сравнительного анализа существующей и экспериментальной страницы, на который изменен один или ряд фрагментов).

Определив группу задач, которые будут решаться в ходе юзабилити-тестирования, вы даете возможность потенциальным клиентам озвучить свои недовольства и пожелания.

Базовый набор инструментов юзабилити-тестирования включает в себя:

- аналитические системы
- web-визор
- эвристический анализ.

К примеру, в процессе юзабилити-тестирования мобильных приложений (такая потребность чаще встает перед маленькими предприятиями), вам желательно заручиться достаточным количеством пользовательских отзывов. Это очень важно, ведь вам нужен ответ на вопрос: почему часть клиентов ушла к другим продавцам, не совершив какие-либо действия именно на вашем ресурсе.

В зависимости от поставленных вами задач по улучшению конверсии сайта и особенностей вашей желаемой целевой аудитории, можно провести:

- классическое юзабилити-тестирование
- онлайн-юзабилити-тестирование.

Кроме того, вам не составит труда понять, – как проводить юзабилити-тестирование. Последовательность ваших действий будет примерно такой:

- выбор сервиса автоматизации (на европейском рынке пользователи обычно предпочитают usertesting.com);

- выбор фокус-группы (к примеру, 20 девушек возрастом от 18 до 28 лет, регулярно заказывающих в интернет-магазинах аксессуары для гаджетов);
- проведение юзабилити-тестирования в течение нескольких дней посредством установки у пользователей софта для захвата и записи экрана.

Хорошо, если разработчики вашего ресурса убедили вас провести юзабилити-тестирование еще до того, как вы окунулись в мир реальных продаж. В этом случае изначально можно быть уверенным в том, что посадочная страница либо страницы каталога, формы выбора товара и оплаты соответствуют устоявшимся представлениям клиентов об удобстве пользования.

Провести тестирование можно и самостоятельно, здесь важен масштаб задач, которые вы перед собой ставите. В качестве фокус-группы, проводя юзабилити-тестирование самостоятельно, вы можете использовать друзей, френдов из соцсетей, людей, зарегистрированных на бирже фрилансеров, однако такой подбор пользователей оправдает себя лишь в случае товаров и услуг, нужных всем без исключения: к примеру, простейших парикмахерских услуг. Но, если вы занимаетесь сопровождением деловых конференций или продаете автозапчасти для салонного транспорта, вам понадобятся реальные потребители ваших услуг, часто прибегающие к механизмам онлайн-покупок и имеющие опыт сотрудничества с разными продавцами.

Следовательно, обратившись к специалистам, вы сможете быть уверенными в том, что для вашего ресурса подберут тематических клиентов либо профессиональных юзабилистов.

Еще один плюс проведения юзабилити-тестирования узкопрофильной web-компанией: по факту завершения анализа вам предоставят не просто отчет о выявленных ошибках, а пакет рекомендаций по улучшению юзабилити ресурса.

Итак, повторим: для тестирования интернет-магазина детских товаров понадобятся молодые родители с доходами не выше среднего, музыкальных дисков – меломаны, готовые платить за лицензионную продукцию, а услуг солярия – жильцы микрорайона, в котором находится ваш салон.

Нельзя ограничивать фокус-группу небольшим количеством пользователей: безусловно, грубые ошибки помогут установить и первые же пять человек, однако глубинные недочеты могут укрыться от их внимания.

Еще одно преимущество курирования процесса профессионалом – специалист точно знает, как проводить юзабилити-тестирование, иначе говоря, он верно поставит задачи и распределит их последовательность.

Прежде чем юзабилити-тестирование будет запущено, необходимо провести подготовительную работу:

- вычленить KPI (ключевые показатели эффективности) вашего ресурса: как правило, это оформление заказа, запрос на обратный звонок, подписка на рассылку;
- вычленение слабых звеньев (например, страниц, с которых пользователь, открыв, сразу уходит) и «холодных этапов», когда, например, пользователь уходит с сайта, дойдя до фильтра товаров;
- написание сценарного плана тестирования (к примеру, составление модулей предполагаемых действий пользователя: «посмотреть каталог – заказать звонок», «посмотреть каталог – отложить в корзину», «посмотреть каталог – оформить покупку»);
- формулирование ТЗ для пользователя фокус-группы, к примеру, «Вы – девушка 20 лет, ищите оригинальный подарок для шефа своей компании стоимостью от столько-то до столько-то».

Как показывает наша практика, по итогам юзабилити-тестирования могут вскрыться совершенно неожиданные для владельца ресурса причины непопулярности сайта или интернет-магазина у целевой группы. К примеру, некоторое количество пользователей обескураживает и отталкивает

необходимостью пройти многоступенчатую регистрацию, прежде чем оформить покупку. Иногда пользователь прерывает процесс выбора товара, столкнувшись с обилием полей, которые необходимо заполнить для перехода в нужный сегмент каталога.

И самое важное: организация юзабилити-тестирования должна охватить и выявить как можно больше проблемных зон ресурса. Лишь в этом случае у специалиста будет достаточное количество исходных данных для составления для вас указаний по улучшению юзабилити

5 Менеджмент тестирования

5.1 Управление тестированием

Для корректного взаимодействия между членами команды тестировщиков и организации грамотного тестирования продукта, используется единая “СУТ” (Система управления тестированием). Системы управления тестированием обычно используются для для планирования ручного тестирования, сбора данных о результатах прохождения чек-листов и тест-кейсов, а также для получения оперативной информации в виде отчетов. Системы управления тестированием помогают оптимизировать процесс тестирования и обеспечивают быстрый доступ к анализу данных, средствам совместной работы и более качественному взаимодействию между несколькими проектными группами.

После старта тестирования проекта члены команды могут взаимодействовать через одну из систем управления тестирования путём создания тест-кейсов, чек-листов, назначая ответственных за прохождение их лиц, что упрощает и улучшает качество взаимодействия лиц, проводящих тестирование в рамках конкретного проекта. При создании или прохождении тестов и чек-листов пользователи могут получить доступ к различным функциям систем управления тестированием, которые автоматизируют данную деятельность и благотворно влияют на скорость и качество её выполнения.

При изучении систем управления тестированием были выделены три наиболее подходящих платформы, предоставляющих бесплатный или относительно бесплатный доступ к своему функционалу.

1) Jira
2) Redmine
3) Trello

5.1.1 Работа в Jira

Изначально JIRA была создана как баг-трекер, но постепенно дорабатывалась, видоизменялась и расширялась. Сейчас это уже не просто баг-трекер. Jira позволяет устанавливать дополнительные плагины для более удобной работы. По сути JIRA очень проста. Есть запросы (issues, или задачи) разных типов, которые могут означать и задачу, и ошибку, и новое требование. У запроса есть атрибуты, можно прикрепить файл, связать его с другим запросом, вести по нему переписку, отслеживать историю изменений. Что очень

важно и полезно, Jira может быть тесно связана с Confluence. Confluence — развитая и удобная Wiki, с помощью которой можно вести базу знаний. Часто в Confluence размещается проектная документация, требования, описания, схемы, инструкции и другая полезная документация и информация по проекту.

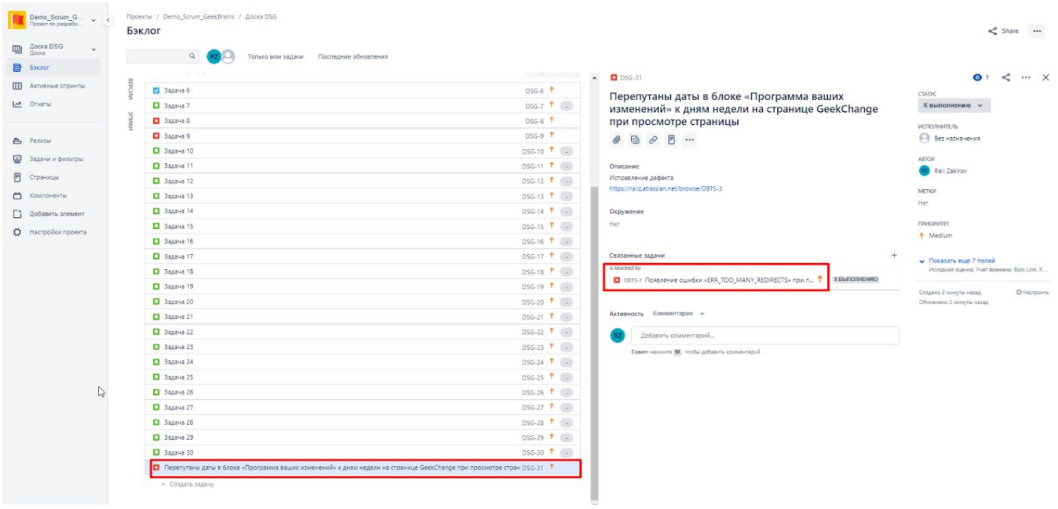
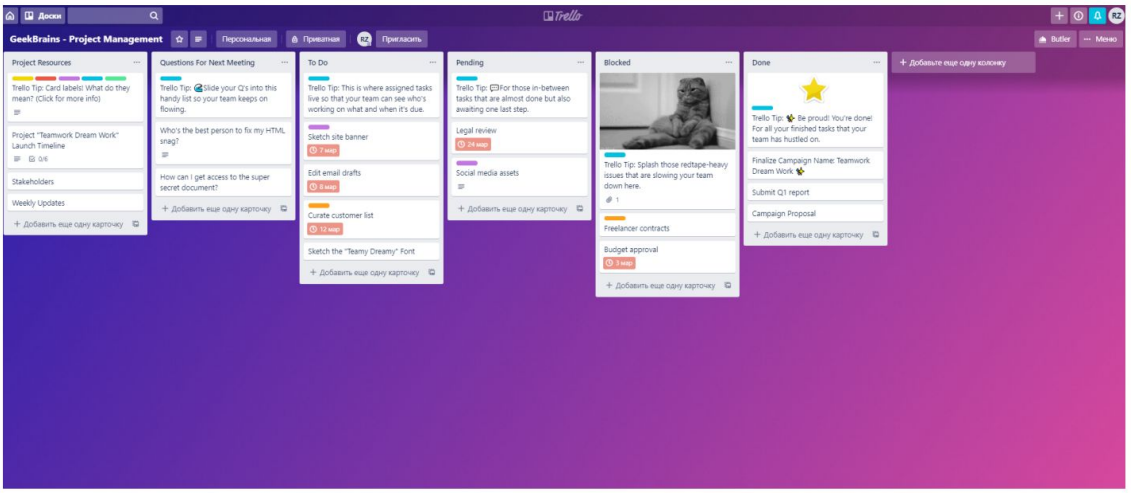


Рисунок - Пример работы со списком задач Jira

5.1.2 Работа в Trello

Это очень простое и доступное веб-приложение для управления задачами. Задачи создаются в виде карточек на доске. История Trello начинается с января 2011 года, когда был представлен первый прототип продукта. 6 В 2017 году проект был выкуплен компанией Atlassian. Существует три версии - веб, мобильная (Android, IOS) и настольная версии. Поддерживаемые браузеры - Chrome, Firefox, Edge, Safari. Приложение доступно на нескольких языках, в том числе и на русском. Приложение очень хорошо подходит для проектов, разрабатываемых по методологии Kanban. В одном приложении можно создавать отдельные доски для нескольких проектов. В начальном варианте использования Trello бесплатно. В платном варианте добавляются дополнительные возможности по администрированию и обеспечению безопасности, ограничению доступа и интеграция с другими внешними приложениями. Чтобы начать работу с веб-приложением, и не устанавливать на свою машину, необходимо зарегистрироваться на сайте.



Пример работы с доской задач Trello

5.1.3 Работа в Redmine

Redmine - Открытое серверное веб-приложение для управления проектами и задачами (в том числе для отслеживания ошибок). Redmine написан на Ruby и представляет собой приложение на основе широко известного веб-фреймворка Ruby on Rails. Распространяется согласно GNU General Public License.

Задачи являются центральным понятием всей системы, описывающим некую задачу, которую необходимо выполнить. У каждой задачи в обязательном порядке есть описание и автор, в обязательном порядке задача привязана к трекеру. Каждая задача имеет статус. Статусы представляют собой отдельную сущность с возможностью определения прав на назначение статуса для различных ролей (например, статус «отклонён» может присвоить только менеджер) или определение актуальности задачи (например, «открыт», «назначен» — актуальные, а «закрыт», «отклонён» — нет). Для каждого проекта отдельно определяются набор этапов разработки и набор категорий задач. Среди других полей интересны также «оценённое время», служащее основой для построения управленческих диаграмм, а также поле выбора наблюдателей за задачей (см. «Получение уведомлений»). К задачам имеется возможность прикреплять файлы (имеется отдельная сущность «Приложение»). Значения других перечислимых свойств (например, приоритетность) хранятся в отдельной общей таблице.

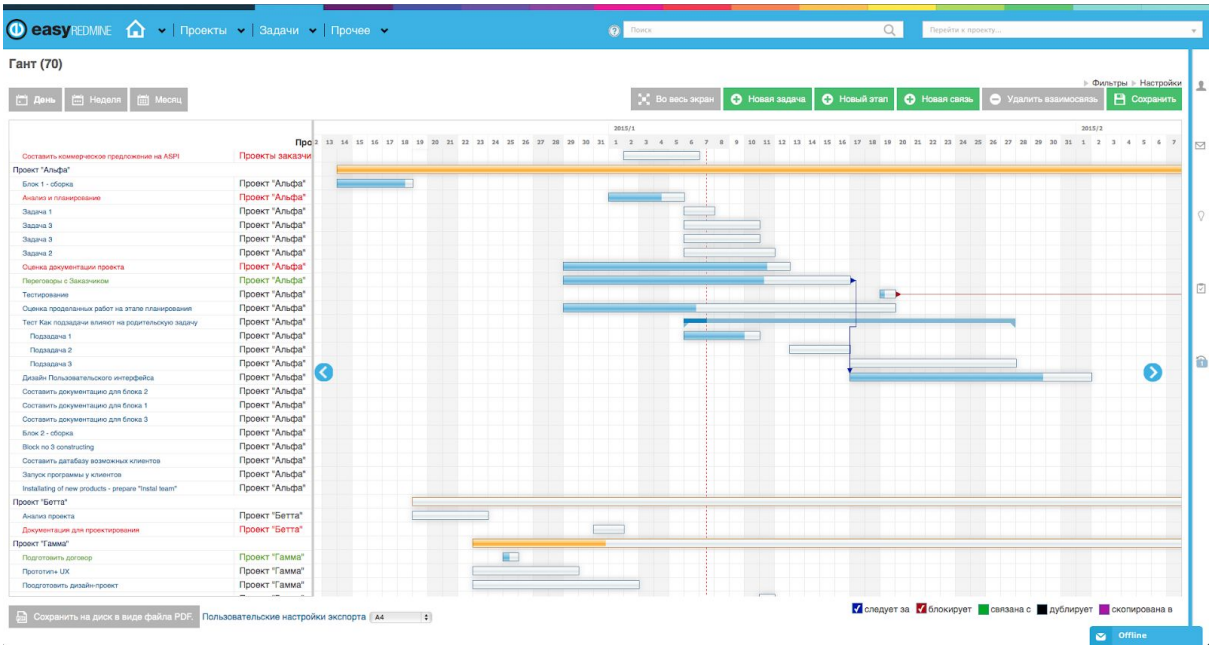


Рисунок - пример работы с доской задач Redmine

5.2 Бизнес-ценность тестирования

На сегодняшний момент сложно отрицать ценность специалистов по тестированию. Основная задача данного специалиста - предотвратить дефекты и, следовательно, обеспечить высокое качество процесса разработки и его результатов. Для более детального рассмотрения функций составим список задач специалиста по тестированию продукта.

- Выявление слабых мест и несоответствия в продукте на всех этапах разработки;
- Помощь в определении требований к проекту;
- Предоставление исчерпывающей информации о качестве продукта;
- Тестирование продукта на протяжении всех фаз жизненного цикла разработки системы (software development lifecycle, SDLC).

Важно отметить, что специалисты по тестированию заинтересованы в том, чтобы сделать любой продукт удобным для пользователя как в плане функциональности, так и в плане дизайна. Для этого они тесно взаимодействуют со всеми членами команды и постоянно обращаются к заданным требованиям.

Множество компаний-разработчиков программного обеспечения работают спринтами — даётся список задач и две недели на их выполнение. Во время каждого спринта они реализуют определённую часть требований к продукту и проходят через все пять стадий тестирования. Важно понимать, что тестирование не подразумевает только проверку каждого способа взаимодействия с продуктом. Конечно, и это тоже, но зачастую с системой можно сделать гораздо больше. Если тестировщик не участвует в процессе разработки, то позже может оказаться, что команда разработчиков сделала что-то, что работает и работает хорошо, но не то, что нужно. Также специалисты помогают сократить время, необходимое для разработки новых тест-кейсов, так как чем раньше разработчики поймут , что и как они собираются тестировать, тем проще будет провести тестирование. Очень важно, чтобы разработчики и тестировщики работали вместе, иначе всё превратится в соревнование «кто найдёт больше багов», что редко приводит к качественным результатам.

Для описания непосредственного влияния тестирования на “Бизнес-сторону” проекта рассмотрим конкретные примеры, где присутствие тестировщика в команде экономически обусловлено.

- **Безопасный бизнес**. У вас есть платёжная система, и она работает нормально. Пользователь платит за услугу и получает её. Однако вы не проверили все возможные случаи, и деньги идут не вам, а на случайный банковский счёт. Такой недочёт может очень дорого обойтись;
- **Экономия денег**. На приведённой ниже диаграмме хорошо видна взаимосвязь между этапами жизненного цикла и затратами. Гораздо дороже исправить ошибку, чем предотвратить её. Исправление одной ошибки может повлечь за собой другие, поэтому количество ваших проблем будет быстро увеличиваться;

- **Защита репутации**. Если выпустить багованный продукт и пользователи не будут довольны работой с ним, в дальнейшем их будет сложно убедить, что проблема решена и они могут снова им пользоваться. Первое впечатление сложно изменить, поэтому предоставьте качественный продукт. Если он не был протестирован вдоль и поперёк, то продукт может работать неправильно или не работать вовсе. Тестирование требует теоретических знаний, поэтому будет сложно обеспечить качество, если вы не профессионал;
- **Контроль процесса**. Если процесс разработки не контролируется и идёт вразрез с установленными требованиями, итоговый продукт может сильно отличаться от запланированного.

5.3 Управление дефектами

Управление дефектами - это систематический процесс выявления и устранения ошибок.

Каждый дефект имеет свой жизненный цикл. Рассмотрим его основные этапы ниже дефекта.

Основные этапы — это:

1. Дефект открыт, когда его обнаружили и создали отчёт о нем.

2. Дефект в работе, когда он назначен на разработчика и разработчик начал работу над ним.

3. Дефект исправлен, когда разработчик внёс изменения (исправление или fix) в программный код и сохранил эти изменения в общем репозитории.

4. Дефект закрыт, когда было проверено, что он действительно исправлен в новой версии продукта.

5. Дефект открыт заново в случае, если ранее исправленный дефект снова был обнаружен в системе.

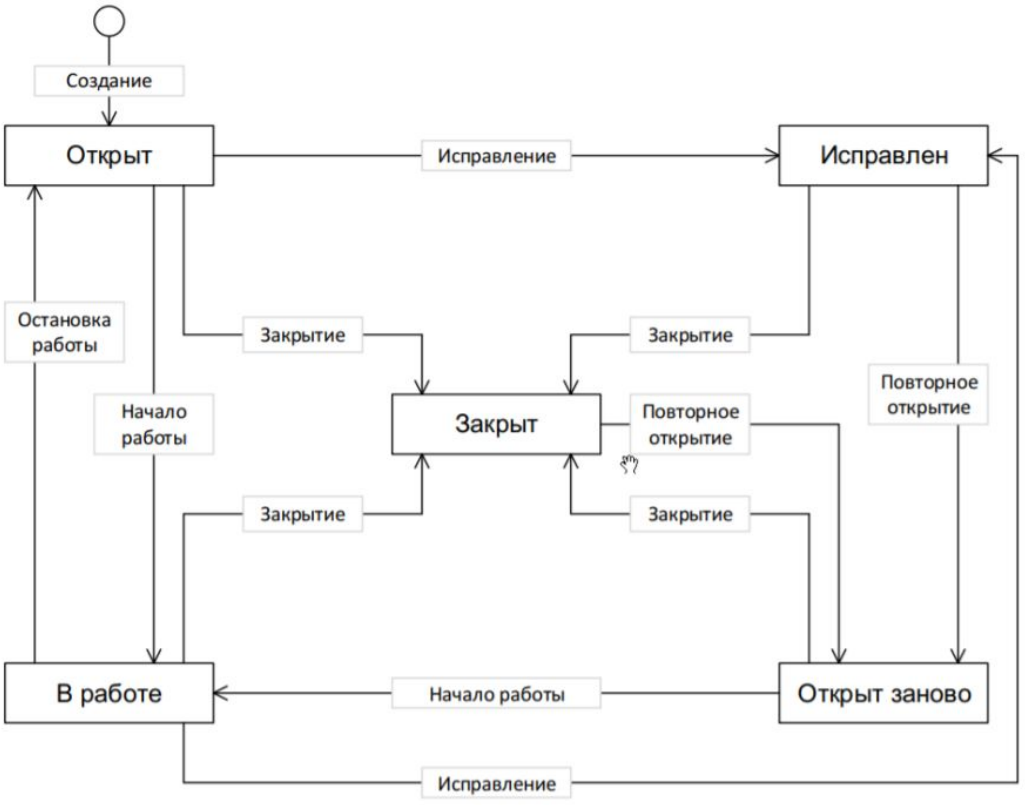


Рисунок - диаграмма иллюстрирующая ЖЦ дефекта

6 ОТЧЕТ ПО ТЕСТИРОВАНИЮ ПРОЕКТА

6.1 Видеозвонки

6.1.1 Тестирование “Видеозвонков”

В части “Тестирование видеозвонков” команде была поставлена задача, протестировать работу видеозвонков в разделе “Сообщения” продавцов и покупателя. Процесс тестирования строился на трех этапах: Написание чек-листа, написание тест-кейсов, описание выявленных дефектов. Также следует отметить, что результаты тестирования также зависят от операционной системы пользователя, поэтому раздел “тест-кейсы” был написан с учетом обеих операционных систем, с наибольшей вероятностью установленных у пользователя.

6.1.2 Чек-листы “видеозвонков”

Написание чек-листов строилось на определении основных функций и объектов подраздела. При анализе раздела были выявлены следующие процессы, требуемые к рассмотрению.

- Проверка перехода на экран видеозвонка
- Проверка разрешения доступа к камере
- Проверка функционала “ожидание ответа”
- Проверка функционала “прием вызова”
- Проверка функционала звонка
- Проверка звонка в фоновом режиме
- Проверка функционала завершенного звонка
- Проверка функционала оценки качества
- Проверка работы камеры
- Проверка времени диалога

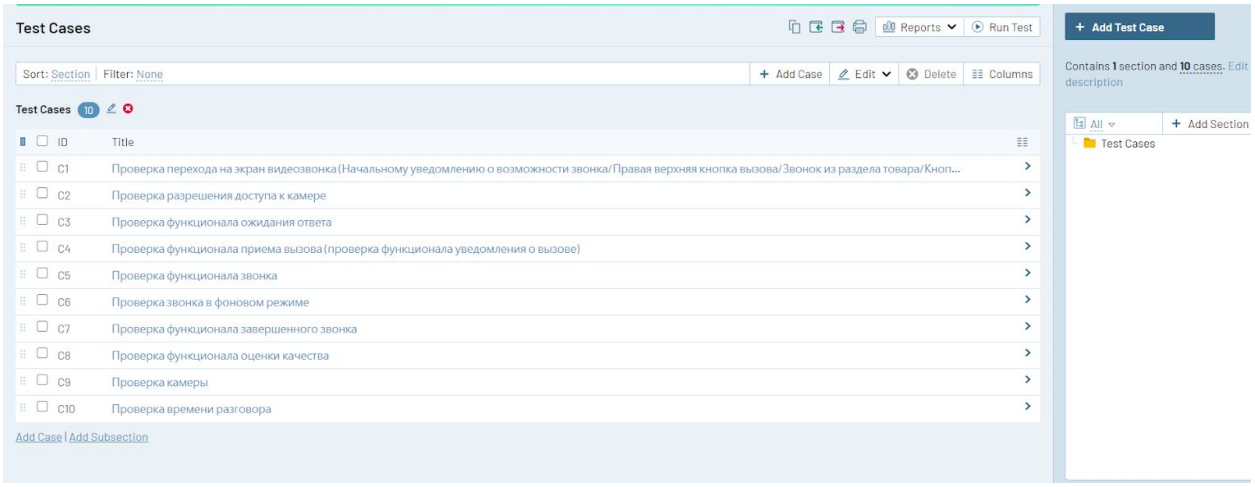


Рисунок - Список чек-листов, составленных в TestRail

6.1.3 Тест-кейсы видеозвонков

Поскольку при написании тест-кейсов, было определено, что в зависимости от операционной системы, приложение по-разному реагирует на один и тот же вид тестирования, было определено составить два тест-кейса, как для Android, так и для IOS систем.
Ниже представлен список функционала, включенного в тест-кейсы исходя из описанных заранее чек-листов.

- Проверка перехода на экран видеозвонка через начальное уведомление о возможности звонка
- Проверка перехода на экран видеозвонка через правую верхнюю кнопку вызова
- Проверка перехода на экран видеозвонка через звонок из раздела товара
- Проверка перехода на экран видеозвонка через ссылку перезвонить
- Проверка отображения заглушки "разрешить доступ к камере" часть 1
- Проверка отображения заглушки "разрешить доступ к камере" часть 2
- Проверка отображения заглушки о "Невозможности звонка" пользователю

- Проверка вкл/выкл микрофона в меню ожидания
- Проверка вкл/выкл камеры в меню ожидания
- Проверка вкл/выкл громкоговорителя в меню ожидания
- Проверка уведомления о том, что Собеседник отклонил вызов
- Проверка завершения звонка из меню ожидания
- Проверка приема вызова через уведомление
- Проверка приема вызова пользователя, находящегося в приложении Юла
- Отображение уведомления о пропущенном звонке
- Отображение уведомления о показе видео собеседником и предложение "включить видео пользователю"
- Проверка вкл/выкл микрофона во время разговора
- Проверка вкл/выкл камеры во время разговора
- Проверка вкл/выкл громкоговорителя во время разговора
- Проверка работы звонка в свернутом режиме
- Проверка завершения звонка
- Переключение режима камеры фронтальная/обычная
- Проверка режима ожидания в фоновом режиме
- Проверка отображения уведомления о завершении звонка
- Проверка кнопки "Написать" при завершении звонка (Данная иконка отображается только при использовании Android)
- Проверка кнопки "перезвонить" при завершенном звонке
- Проверка кнопки "отменить" при завершенном звонке
- Проверка оценки связи при завершенном звонке (При использовании Android данного уведомления не было)
- Проверка правильности отсчета длительности звонка

Рисунок - Список тест-кейсов , составленных в TestRail для Android

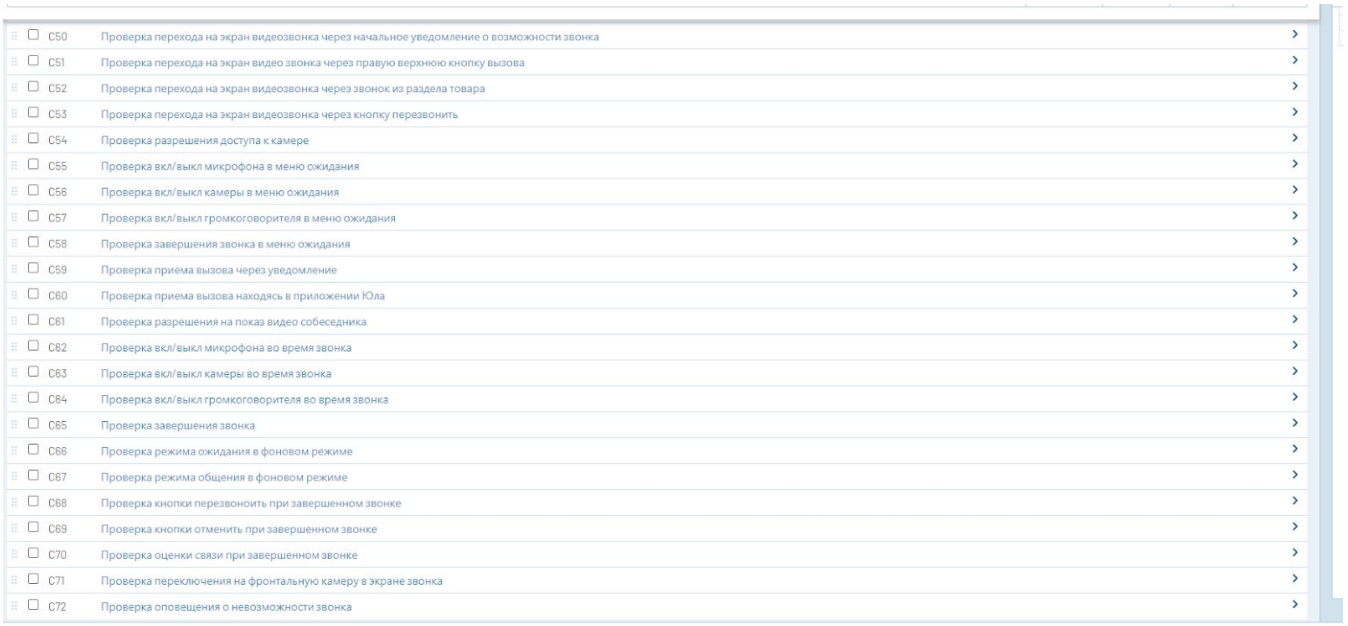


Рисунок - Список тест-кейсов , составленных в TestRail для IOS

Пример описания тест-кейса на пункте “Проверка отображения заглушки о "Невозможности звонка" пользователю” представлен на рисунке ниже.

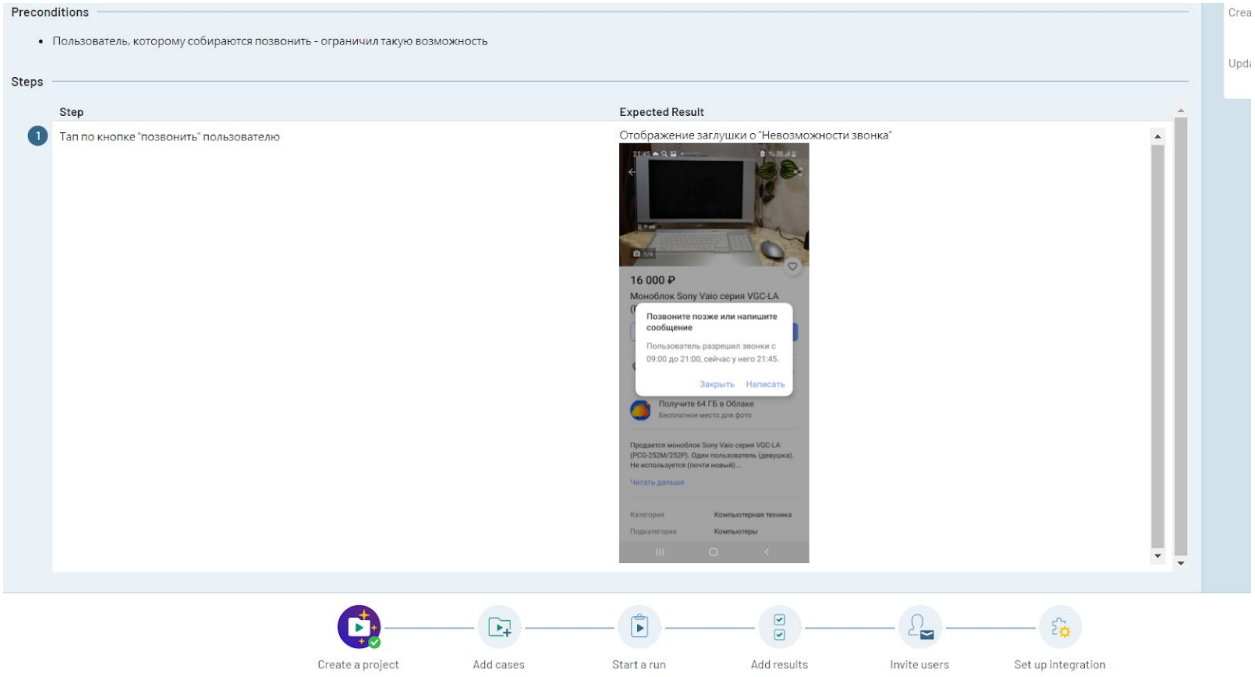


Рисунок - Пример описания тест-кейса на пункте “Проверка отображения заглушки о "Невозможности звонка"

6.1.4 Найденные дефекты для раздела “видеозвонки”

В ходе проведения тест-кейсов был обнаружен баг, существенно влияющий на работу приложения. Его краткое описание будет представлено в формате плана, для каждого найденного дефекта.

Тема : Некорректная работа динамика при включенном громкоговорителе
Подробное описание : Во время звонка, при использовании телефона на системе IOS, в момент, когда кнопка громкоговорителя включена, не представляется возможным отключить громкоговоритель вплоть до завершения звонка, по сути кнопка громкого динамика становится бесполезной.
Шаги для обнаружения бага :

1) Один из собеседников должен держать связь с телефоном на IOS платформе

2) Созвониться с собеседником
3) Нажать кнопку включения динамика
4) Попытаться отключить кнопку громкоговорителя - момент возникновения дефекта.

Приоритет дефекта: Значительный дефект (major)

6.2 Аудиозвонки

6.2.1 Тестирование “Аудиозвонков”

В части “Тестирование аудиозвонков” команде была поставлена задача, протестировать работу аудиозвонков в разделе “Сообщения” продавцов и покупателя. Процесс тестирования строился на трех этапах: Написание чек-листа, написание тест-кейсов, описание выявленных дефектов. Также следует отметить, что результаты тестирования также зависят от операционной системы пользователя, поэтому раздел “тест-кейсы” был написан с учетом таких операционных систем, как Andoid и IOS.

6.2.2 Чек-листы аудизвонков

Написание чек-листов строилось на определении основных функций и объектов подраздела. При анализе раздела были выявлены следующие процессы, требуемые к рассмотрению.

- Проверка аудиозвонка, при условии,что продавец разрешил функцию звонка,как по номеру телефону,так и через приложение,при этом время звонка совпадает с выбранным.
- Проверка аудиозвонка, при условии,что продавце запретил звонить в определенное время.
- Проверка аудизвонка,при условии,что продавец разрешил звонок исключительно через приложение
- Проверка аудизвонка в диалоге
- Проверка аудизвонка,исключительно через приложение
- Проверка ожидани аудизвонка
- Проверка аудиозвонка,при заблокированном телефоне

6.2.3 Тест-кейсы аудизвонков

Ниже представлен список функционала, включенного в тест-кейсы исходя из описанных заранее чек-листов.

	Настройка звонков	Тестирование процесса	Шаги	Ожидаемый результат	Фактический результат
2	Разрешить звонить мне+ По номеру телефону+ Через Юлу+ с 03-00 до 03-00	Аудиозвонок по объявлению через Юлу	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок через юлу	Вариант "Бесплатно через Юлу" присутствует. Звонок проходит	Вариант "Бесплатно через Юлу" присутствует. Звонок проходит.
3	Разрешить звонить мне+ По номеру телефону+ Через Юлу+ с 03-00 до 03-00	Аудиозвонок по объявлению через телефон	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок по нмеру телефона	Вариант "Номер телефона" присутствует. Звонок проходит.	Вариант "Номер телефона" присутствует. Приходит сообщение, что номер просматривали. Звонок проходит.
4	Разрешить звонить мне- с 03-00 до 03-00	Аудиозвонок по объявлению через Юлу	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок через юлу	Вариант "Бесплатно через Юлу" отсутствует	Вариант "Бесплатно через Юлу" отсутствует
5	Разрешить звонить мне- с 03-00 до 03-00	Аудиозвонок по объявлению через телефон	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок по номеру телефона	Вариант "Номер телефона" отсутствует	Вариант "Номер телефона" отсутствует
6	Разрешить звонить+ По номеру телефону- Через Юлу+ с 03-00 до 03-00	Аудиозвонок по объявлению через Юлу	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок через юлу	Вариант "Бесплатно через Юлу" присутствует. Звонок проходит	Звонок проходит, но начинается сразу после нажатия на кнопку Позвонить, без предоставления выбора.
7	Разрешить звонить+ По номеру телефону- Через Юлу+ с 03-00 до 03-00	Аудиозвонок по объявлению через телефон	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок по номеру телефона	Вариант "Номер телефона" отсутствует	Вариант "Номер телефона" отсутствует
8	Разрешить звонить+ По номеру телефону+ Через Юлу- с 03-00 до 03-00	Аудиозвонок по объявлению через Юлу	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок через юлу	Вариант "Бесплатно через Юлу" отсутствует	Вариант "Бесплатно через Юлу" отсутствует
9	Разрешить звонить+ По номеру телефону+ Через Юлу- с 03-00 до 03-00	Аудиозвонок по объявлению через телефон	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок по номеру телефона	Вариант "Номер телефона" присутствует. Звонок проходит.	Вариант "Номер телефона" присутствует. Приходит сообщение, что номер просматривали. Звонок проходит.
10	Разрешить звонить мне+ По номеру телефону+ Через Юлу+ Звонок в запрещённое время	Аудиозвонок по объявлению через Юлу в неудобное время	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок через юлу	Вариант "Бесплатно через Юлу" отсутствует	Вариант "Бесплатно через Юлу" отсутствует. У покупателя всплывает окно с информацией, когда можно дозвониться.
11	Разрешить звонить мне+ По номеру телефону+ Через Юлу+ Звонок в запрещённое время	Аудиозвонок по объявлению через телефон в неудобное время	1)Зайти на объявление 2)Кнопка позвонить 3)Выбрать звонок по номеру телефона	Вариант "Номер телефона" отсутствует	Вариант "Номер телефона" отсутствует. У покупателя всплывает окно с информацией, когда можно дозвониться.

Рисунок - Список тест-кейсов

6.2.4 Найденные дефекты для раздела аудиозвонки

В ходе проведения тест-кейсов был обнаружен баг, существенно влияющий на работу приложения. Его краткое описание будет представлено в формате плана, для каждого найденного дефекта.

Тема : Некорректная работа приложения при настройки аудиозвонка.

Подробное описание : Баг с кэшированием. Для того чтобы настройки применились, необходимо было каждый раз перезаходить в приложение. И, соответственно, без перезапуска приложения все настройки были как при его последнем запуске. Отключив разрешение звонить, пользователь всё равно может получить звонок, если он или его будущий собеседник не вышел из приложения.

Шаги для обнаружения бага :

1. Зайти в настройки разрешения звонка
2. Отключить аудиозвонок
3. Не выходить из приложения
4. Позвонить этому пользователю

Приоритет дефекта: Значительный дефект (major)

ЗАКЛЮЧЕНИЕ

В рамках проектной деятельности за семестр командой тестировщиков в составе студентов Московского политехнического университета было проведено тестирование мобильного приложения “Юла” от разработчиков «Майл.ру» с предоставлением тестовой документации и описанием выявленных дефектов. Тестовая документация включала в себя чек-листы и тест-кейсы по тестированию аудио-, видео звонков и звонков внутри приложения при помощи p2p соединения, составленные тремя группами тестировщиков соответственно:

- группа 1: Селиванов С.Д., студент группы 171-372, Белов В.А. , студент группы 171-371;
- группа 2: Колпаков А.А., студент группы 171-372, Камашева Ю.О., студент группы 171-371;
- группа 3: Ласкин В.Д., студента ФИТ, группы 191-331, Малькина А.А., студентка ФИТ, группы 181-321.

В процессе работы были изучены различные виды тестирования программного обеспечения, в частности тестирование API, тестирование WEB, нагрузочное тестирование, автоматизированное тестирование, ручное тестирование, юзабилити тестирование. При этом на практике были опробованы ручное и юзабилити тестирования.

Была проанализирована работа систем управления тестированием (менеджмента тестирования), которые используются для хранения информации, для получения информации в виде отчета о стадии тестирования и качестве тестируемого продукта. Следует отметить также высокую бизнес-ценность тестирования в современном профессиональном мире и существенную важность управления дефектами.

Статистические данные проведенных группами 1, 2, 3 тестирований представлены в пункте 6 данного отчета по проектной деятельности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Борис Бейзер «Тестирование черного ящика» от 04.01.2004 под редакцией издательства «Питер» https://codernet.ru/books/QA/testirovanie_chernogo_yashchika_boris_beyzer/
2. Гленфорд Майерс, Том Баджетт, Кори Сандлер «Искусство тестирования программ» от 22.10.2009 под редакцией издательства "Финансы и статистика"
3. Канер Сэм, Фолк Джек, Нгуен Енг Кек «Тестирование программного обеспечения» от 13.05.2001 под редакцией издательства «ДиаСофт». https://codernet.ru/books/QA/testirovanie_programmnogo_obespecheniya_sem_kaner_2001/
4. Лиза Криспин, Джанет Грегори «Гибкое тестирование» от 12.11.2016 под редакцией издательства «Signature Series»
5. Рекс Блэк «Ключевые процессы тестирования» от 17.04.2014 под редакцией издательства «Лори» https://codernet.ru/books/QA/klyuchevye_protsessy_testirovaniya_reks_blek/
6. Роберт Калбертсон, Крис Браун, Гэри Кобб — «Быстрое тестирование» от 16.02.2007 под редакцией издательства «Вильямс» https://codernet.ru/books/QA/bystroe_testirovanie_robert_kalbertson/
7. Святослав Куликов «Тестирование программного обеспечения. Базовый курс» от 29.09.2015 под редакцией «Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International» https://fktpm.ru/file/113-svyatoslav-kulikov-testirovanie-po-bazovyi-kurs.pdf
8. Рекс Блэк «Ключевые процессы тестирования» от 17.04.2014 под редакцией издательства «Лори» https://codernet.ru/books/QA/klyuchevye_protsessy_testirovaniya_reks_blek/
9. Список обучающих лекций по работе над тестирование под редакцией «Lime Lab»