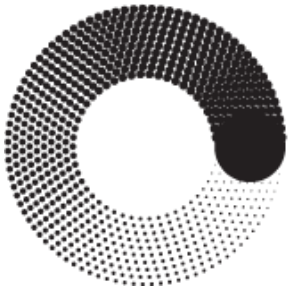
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
(ФГБОУ ВПО МПУ)



МОСКОВСКИЙ ПОЛИТЕХ

Пояснительная записка по дисциплине
«Проектная деятельность»

По теме: «Лаборатория тестирования. Тестирование приложения “Пассажирам”.»

Куратор проекта: _____ / Яковлев Станислав Игоревич ,Преподаватель /
подпись ФИО, уч. звание и степень

Студент: _____ / Гетун Дмитрий Юрьевич , 181-352/
подпись ФИО, группа

Москва, 2021

Оглавление

Общее задание, план работы

Протестировать iOS версию приложения “РЖД. Пассажирам”, выявить недостатки и дефекты, подготовить чек-листы и тест-кейсы, а также составить отчеты о дефектах при наличии.

Вид тестирования: ручное тестирование

Задача	Описание	Кол-во часов
Изучить функционал приложения	Изучить основной функционал приложения	3
Изучить виды тестирования и техники тест-дизайна, составить отчет		10
Разработать чек-листы	Разработать чек-листы для проверки функционала приложения	5
Разработать тест-кейсы	Разработать тест-кейсы для проверки функционала приложения	25
Протестировать приложение	Протестировать приложение с использованием полученных знаний и разработанных тест-кейсов	20

Оформить отчеты о найденных дефектах	Оформить отчеты о дефектах в соответствии с правилами	2

Индивидуальные планы участников

Гетун Д.Ю. 181-352. Составить чек-листы и тест-кейсы, протестировать приложение и составить отчеты о дефектах

Содержательные главы

Виды тестирования

По виду тестирование делится на следующие группы:

1. По цели тестирования
    - Функциональное
    - Нефункциональное
    - Связанные с изменениями в коде
2. По исполнителям тестирования
    - Альфа-тестирование
    - Бета-тестирование
3. По уровню функционального тестирования
    - Дымовое
    - Критического пути
    - Расширенное
4. По принципам работы с приложением
    - Позитивное
    - Негативное
5. По запуску кода на исполнение
    - Статическое
    - Динамическое
6. По доступу к коду и архитектуре приложения
    - Метод белого ящика
    - Метод серого ящика
    - Метод черного ящика
7. По уровню детализации приложения
    - Модульное
    - Интеграционное
    - Системное

8. По степени автоматизации
    - Ручное
    - Автоматизированное

Рассмотрим каждое из них подробнее:

1. По цели Тестирования

а) Функциональное

Задача функционального тестирования – установить соответствие разработанного ПО исходным функциональным требованиям заказчика, то есть способность системы решать задачи, необходимые пользователям.

Тестирование может проводиться:

- На основе функциональных требований, указанных в спецификации требований. При этом для тестирования создаются тестовые случаи (testcases), составление которых учитывает приоритетность функций ПО. Таким образом, мы можем убедиться, что все функции разрабатываемого продукта работают корректно при различных типах входных данных, их комбинаций, количества и т.д.
- На основе бизнес-процессов, которые должны обеспечить приложение. Нас будет интересовать корректность выполняемых функций, с точки зрения сценариев использования системы (usecases).

Преимущества:

- Полностью имитирует фактическое использование системы
- Позволяет своевременно выявить системные ошибки
- Исправление ошибок на более раннем этапе жизненного цикла ПО

б) Нефункциональное

Нефункциональное тестирование – тестирование свойств, которые не относятся к функциональности системы. Данные свойства характеризуются

нефункциональными требованиями, которые характеризуют продукт с таких сторон, как:

- Надежность – реакция системы на непредвиденные ситуации
- Производительность – работоспособность системы под разными нагрузками
- Удобство – исследование удобности работы с приложением с точки зрения пользователя
- Масштабируемость – Требования к горизонтальному или вертикальному масштабированию приложения
- Портируемость – переносимость приложения на различные платформы

в) Связанные с изменениями в коде

После проведения необходимых изменений, таких как исправление бага/дефекта, ПО должно быть перетестировано для подтверждения того факта, что проблема была действительно решена.

Тестирования, которые должно быть проведены для подтверждения работоспособности приложения или правильности исправления дефекта:

- Дымовое тестирование
- Регрессионное тестирование
- Тестирование сборки

2. По исполнителям тестирования

а) Альфа-тестирование

Один из видов приемочного тестирования. Осуществляется контролируемым образом в среде тестирования без привлечения внешних пользователей, за счет внутренних сил компании.

Выполняется для имитации поведения в режиме реального времени в соответствии с использованием продукта конечными пользователями на рынке.

На этом этапе основное внимание уделяется:

- Обнаружению ошибок
- Вопросам юзабилити
- Различию в характеристиках
- Проблемам совместимости/ взаимодействия

Данное тестирование выполняется, когда продукт готов на 70% - 90%

б) Бета-тестирование

Интенсивное тестирование почти готовой версии продукта с целью выявления максимального числа ошибок в его работе.

Предполагает привлечение добровольцев из числа будущих пользователей, которым доступна предварительная версия продукта.

Бета-версия не является финальной версией продукта.

Цели:

- Получить полный обзор истинного опыта, накопленного конечными пользователями при работе с продуктом.
- Использование большого числа платформ для тестирования
- Выявить скрытые ошибки и пробелы в конечном продукте

Выполняется, когда продукт готов на 90% - 95%.

Этапы:

- Планирование
- Набор участников
- Запуск продукта
- Сбор и оценка отзывов
- Закрытие

3. По уровню функционального тестирования

а) Дымовое

Короткий цикл тестов, выполняемый для подтверждения того, что после сборки кода устанавливаемое приложение стартует и выполняет основные функции.

Вывод о работоспособности основных функций делается на основании результатов поверхностного тестирования наиболее важных модулей приложения на предмет возможности выполнения требуемых задач и наличия быстронаходимых критических и блокирующих дефектов. В случае успешного прохождения теста, приложение передается для проведения полного цикла тестирования. В случае провала, приложение уходит на доработку.

б) Критического пути

Тип тестовых испытаний, во время которого значимые элементы и функции приложения проверяются на предмет правильности работы при стандартном их использовании.

На данном уровне тестирования проверяется основная масса требований к продукту. Например, возможность набора текста, вставка картинок, возможность войти на сайт, создать запись и т.д.

Тест критического пути является одним из самых распространенных видов функционального тестирования. Частота проведения данного тестирования обусловлена в первую очередь необходимостью периодической проверки всего приложения в сжатые сроки. А также позволяет выявить самые быстро находимые дефекты и исправить приложение в более сжатые сроки.

в) Расширенное

Тестирование, при котором проверяется нестандартное использование программного продукта, границы переполнения массивов данных, ввод специальных символов, нелогичное кликанье по кнопкам, открыть одно окно и закрыть предыдущее и т.д.

Обычно данный вид тестирования проводится при наличии времени и ресурсов. Тест-кейсы для него составляются аналогично тест-кейсам основного функционала, но их обычно меньше всего и они могут всегда дополняться.

4. По принципам работы с приложением

а) Позитивное

Тестирование с применением сценариев, которые соответствуют нормальному (штатному) поведению системы. С его помощью мы можем определить, что система делает то, для чего была создана.

Позитивное тестирование направлено на проверку работы системы с теми типами данных, для которых она разрабатывалась.

б) Негативное

Тестирование с применением сценариев, которые являются нештатными для системы.

Негативное тестирование направлено на проверку устойчивости системы к различным воздействиям, валидации неверных данных, обработки исключительных ситуаций.

5. По запуску кода на исполнение

а) Статическое

Тип тестирования, который предполагает, что программный код во время тестирования не будет выполняться.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является частью процесса верификации.

б) Динамическое

Тип тестирования, который предполагает запуск программного кода. Таким образом, анализируется поведение программы во время её работы.

При этом, может выполняться проверка внешних параметров работы программы: загрузка процессора, использование памяти, время отклика и т.д.

Является частью процесса валидации.

6. По доступу к коду и архитектуре приложения

а) Метод белого ящика

Метод тестирования ПО, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику.

Входные значения выбираются на основании знания кода, результат их обработки известен.

Тестирование белого ящика – углубление во внутреннее устройство системы, за пределы её внешних интерфейсов.

Преимущества:

- Тестирование может производиться на ранних этапах
- Покрытие большого количества путей выполнения программы

Недостатки:

- Необходимо большое количество знаний

- Поддержка тестовых скриптов может оказаться накладной при частых изменениях программы.

б) Метод черного ящика

Техника тестирования, основанная на работе исключительно с внешними интерфейсами системы.

Тестирование методом черного ящика не предполагает знаний внутреннего устройства компонента или системы.

Целью этой техники является поиск ошибок в таких категориях:

- Неправильно реализованные или недостающие функции
- Ошибки интерфейса
- Ошибки в структурах данных
- Ошибки поведения или недостаточная производительность системы

в) Метод серого ящика

Метод тестирования, который предполагает комбинацию белого и черного ящика. То есть, внутреннее устройство программы нам известно лишь частично.

Предполагается, например, доступ к внутренней структуре и алгоритмам работы ПО для написания максимально эффективных тест-кейсов, но само тестирование проводится с помощью техники черного ящика, то есть, с позиции пользователя.

7. По уровню детализации приложения

а) Модульное

Тестирование каждой функциональности приложения отдельно, в искусственно созданной среде. Для этого используются драйверы и заглушки.

Драйвер – модуль теста, который выполняет тестируемый нами элемент.

Заглушка – часть программы, которая симулирует обмен данными с тестируемым компонентом, выполняет имитацию рабочей системы.

Заглушки нужны для:

- Имитирования недостающих компонентов для работы данного элемента
- Подачи или возвращения модулю определенного значения
- Воссоздания определенных ситуаций

б) Интеграционное

Тип тестирования, при котором программные модули объединяются логически и тестируются как группа.

Целью является выявление багов при взаимодействии между программными модулями и проверка обмена данными между этими самыми модулями.

Интеграционное тестирование отличается от других тем, что сосредоточено в основном на интерфейсах и потоке данных между модулями. Здесь приоритет проверки присваивается интегрирующим ссылкам, а не функциям блока, которые уже проверены.

в) Системное

Тестирование ПО, выполняемое на полной, интегрированной системе с целью проверки соответствия системы исходным требованиям, как функциональным, так и не функциональным.

Системное тестирование выполняется методом “Черного ящика” т.к. проверяемое множество является внешними сущностями, которые не требуют взаимодействия с внутренним устройством программы.

Выполнять рекомендуется в окружении, максимально приближенном к окружению конечного пользователя.

Можно выделить два подхода к системному тестированию:

- На базе требований – в соответствии с функциональными или нефункциональными требованиями, для каждого из которых пишется тест-кейс.
- На базе случаев использования – в соответствии с вариантами использования продукта, на основе которых создаются юзер-кейсы.

8. По степени автоматизации

а) Ручное

При ручном тестировании тестировщики вручную выполняют тесты, не используя никаких средств автоматизации, моделируя действия реальных пользователей.

Самый низкоуровневый и простой тип тестирования, не требующих большого количества дополнительных знаний.

б) Автоматизированное

Предполагает использование специального ПО для контроля выполнения тестов и сравнения ожидаемого и фактического работы программы.

После создания автоматизированных тестов, их можно в любой момент запустить снова, причем запускаются и выполняются они быстро и точно.

Техники тест-дизайна

1. Тестирование классами Эквивалентности

   Тестовые данные разбиваются на определенные классы допустимых значений. В рамках каждого класса выполнение теста с любым значением тестовых данных приводит к эквивалентному результату. После определения классов необходимо выполнить хотя бы один тест в каждом классе.

Продемонстрируем это на конкретном примере. Представим, что мы тестируем модуль для HR, который определяет возможность принятия на работу кандидата в зависимости от его возраста. Установлены следующие условия отбора:

- при возрасте от 0 до 16 лет – не нанимать;
- при возрасте от 16 до 18 лет – можно нанять только на part time;
- при возрасте от 18 до 55 лет – можно нанять на full time;
- при возрасте от 55 до 99 лет – не нанимать.

Стоит ли проверять каждое значение? Более разумным видится тестирование диапазона каждого условия. Собственно, это и есть наши классы эквивалентности:

- класс эквивалентности NO: 0-16;
- класс эквивалентности PART: 17-18;
- класс эквивалентности FULL: 19-55;
- класс эквивалентности NO: 56-99.

Как уже было указано, после определения классов эквивалентности мы должны выполнить тест с любым значением из диапазона класса. Таким образом, у нас остается только 4 позитивных тест-кейса вместо первоначальных 100 (0-99), например:

- 10 – не нанимать;
- 17 – нанимать на неполный день;
- 40 – нанимать на полный день;
- 80 – не нанимать.

2. Тестирование граничных значений

   Эта техника основана на том факте, что одним из самых слабых мест любого программного продукта является область граничных значений. Для начала выбираются диапазоны значений – как правило, это классы эквивалентности. Затем определяются границы диапазонов. На каждую из границ создается 3 тест-кейса: первый проверяет значение границы, второй – значение ниже границы, третий – значение выше границы.

   Вернемся к примеру, рассмотренному нами в технике классов эквивалентности:

   - при возрасте от 0 до 16 лет – не нанимать;
   - при возрасте от 16 до 18 лет – можно нанять только на part time;
   - при возрасте от 18 до 55 лет – можно нанять на full time;
   - при возрасте от 55 до 99 лет – не нанимать.

Таким образом, набор значений, для которых будут составлены тесты, будет выглядеть так: {-1, 0, 1}, {15, 16, 17}, {17, 18, 19}, {54, 55, 56}, {98, 99, 100}.

3. Таблица принятия решений

Таблицы решений – это удобный инструмент для фиксирования требований и описания функциональности приложения. Таблицами очень удобно описывать бизнес-логику приложения, и они могут служить отличной основой для создания тест-кейсов.

Таблицы решений описывают логику приложения, основываясь на условиях системы, характеризующих ее состояния. Каждая таблица должна описывать одно состояние системы. Шаблон таблицы решений следующий:

	Тест 1	Тест 2	…	Тест p
Условия				
Условие-1				
…				
Условие-n				
Действия				
Действие-1				
…				
Действие-m				

4. Тестирование состояний и переходов

Система переходит в то или иное состояние в зависимости от того, какие операции над нею выполняются. Для наглядности возьмем классический пример покупки авиабилетов:

Все начинается с точки входа. Мы (клиенты) предоставляем авиакомпании информацию для бронирования. Служащий авиакомпании является интерфейсом между нами и системой бронирования авиабилетов. Он использует предоставленную нами информацию для создания бронирования. После этого наше бронирование находится в состоянии «Создано». После создания бронирования система также запускает таймер. Если время таймера истекает, а забронированный билет еще не оплачен, то система автоматически снимает бронь.

Каждое действие, выполненное над билетом, и соответствующее состояние (отмена бронирования пользователем, оплата билета, получение билета на руки, и т. д.) отображаются в блок-схеме. На основании полученной схемы составляется набор тестов, в котором хотя бы раз проверяются все переходы.

Некоторым исследователям представляется более удобным свести весь процесс в таблицу состояний и переходов. Конечно, таблица не так наглядна, как схема, но зато она получается более полной и систематизированной, так как определяет все возможные State-Transition варианты, а не только валидные.

5. Метод парного тестирования

Метод парного тестирования основан на следующей идее: подавляющее большинство багов выявляется тестом, проверяющим либо один параметр, либо сочетание двух. Ошибки, причиной

которых явились комбинации трех и более параметров, как правило, значительно менее критичны.

Допустим, что мы имеем систему, которая зависит от нескольких входных параметров. Да, мы можем проверить все возможные варианты сочетания этих параметров. Но даже для случая, когда каждый из 10 параметров имеет всего два значения (Вкл/Выкл), мы получаем $2^{10} = 1024$ комбинаций! Используя метод парного тестирования, мы не тестируем все возможные сочетания входных параметров, а составляем тестовые наборы так, чтобы каждое значение параметра хотя бы один раз сочеталось с каждым значением остальных тестируемых параметров. Таким образом, метод существенно сокращает количество тестов, а значит, и время тестирования.

Но изюминка метода не в том, чтобы перебрать все возможные пары параметров, а в том, чтобы подобрать пары, обеспечивающие максимально эффективную проверку при минимальном количестве выполняемых тестов. С этой задачей помогают справиться математические методы, называемые ортогональными таблицами.

6. Доменный анализ

Это техника основана на разбиении диапазона возможных значений переменной (или переменных) на поддиапазоны (или домены), с последующим выбором одного или нескольких значений из каждого домена для тестирования. Во многом доменное тестирование пересекается с известными нам техниками разбиения на классы эквивалентности и анализа граничных значений. Но доменное тестирование не ограничивается перечисленными техниками. Оно включает в себя как анализ зависимостей между переменными, так и поиск тех значений переменных, которые несут в себе большой риск (не только на границах).

7. Сценарий использования

   Use Case описывает сценарий взаимодействия двух и более участников (как правило – пользователя и системы). Пользователем может выступать как человек, так и другая система. Для тестировщиков Use Case являются отличной базой для формирования тестовых сценариев (тест-кейсов), так как они описывают, в каком контексте должно производиться каждое действие пользователя. Use Case, по умолчанию, являются тестируемыми требованиями, так как в них всегда указана цель, которой нужно достигнуть, и шаги, которые надо для этого воспроизвести.

Результаты

Тест-кейсы и чек-листы представлены по ссылке ниже:

Address: https://oi2828.testrail.io/

Username: dmgetun@gmail.com

Password: vkqCiVuZ5tXQs5qaj0aR

GitHub:
https://github.com/DmGetun/PD

Отчеты о дефектах

Английская локализация

Предусловие

1. Открыта главная страница приложения

Шаги воспроизведения

1. Нажать значок “Меню” в левом верхнем углу экрана.
2. В раскрывшемся меню выбрать пункт “Настройки”.
3. На открывшейся странице выбрать пункт “Язык приложения”.
4. Выбрать английский язык.
5. Нажать кнопку “continue”.

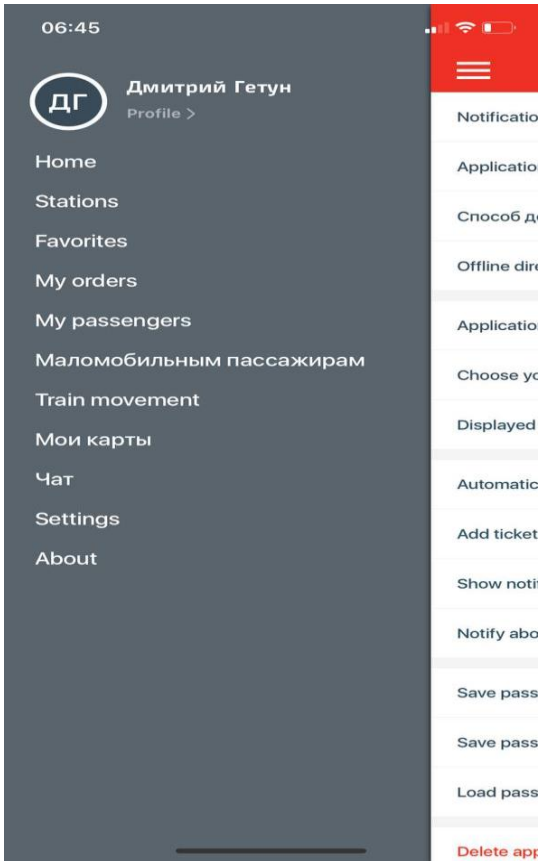
Ожидаемый результат

Все элементы приложения переведены на английский язык.

Фактический результат

Элементы приложения переведены на русский и английский языки одновременно.

Вложение



Китайская локализация

Предусловие

2. Открыта главная страница приложения

Шаги воспроизведения

6. Нажать значок “Меню” в левом верхнем углу экрана.
7. В раскрывшемся меню выбрать пункт “Настройки”.
8. На открывшейся странице выбрать пункт “Язык приложения”.
9. Выбрать китайский язык.
10. Нажать кнопку снизу экрана

Ожидаемый результат

Все элементы приложения переведены на китайский язык.

Фактический результат

Элементы приложения переведены на русский, английский и китайский языки одновременно.

Вложение