***СП***

***Тестирование 5.09***

Silenium

Postman

Библио-Тестирование

Главная и основная цель тестирования, как трудовой деятельности – повышение качества продукта.

Процесс тестирования, в рамках обозначенной цели заключается в поиске, локализации и корректной описании бага.

Согласно теории тестирования – исчерпывающее тестирование невозможно. Невозможно в приложении найти все баги за конечное время. Повышение качества продукта заключается в достижении приемлемой, стабильной работы приложения, устраивающей заказчика и пользователя.

Ресурсы (время, технические ресурсы(номенклатура устройств или программных компонент, которые нам доступны), человеческий ресурс), выделяемые на тестирование, всегда ограничены.

Основная проблема тестирования – необходимость достичь цели, в заданных ограниченных условиях.

Инструменты тестировщика:

Любой таск-менеджер (Редмайн и жира) в этих системах тестировщик фиксирует результат своей работы.

Средства автоматизации – selenium, postman, jemkins и др.

Основным инструментом является инструмент фиксации найденных им багов.

Этапы разработки любого проекта

1. Концепция
2. Договор
3. Техническое задание
4. Дизайн
5. Разработка
6. Стабилизация
7. Релиз
8. Поддержка
9. Закрытие проекта

Тестировщик начинает свою работу на этапе составления тз. Тестировщик тестирует техническое задание на полноту и на непротиворечивость. Под полнотой подразумевается свойство при котором исключается двоякое толкование.

Когда дизайн нарисован, тестировщик проверяет его соответствие требованиям и его полную реализацию.

На этапе разработки тестировщик определяет стратегию и план тестирования, пишет чек-листы, разрабатывает тест-кейсы, может написать автоматизацию на основание требований.

Стратегия тестирования отвечает на вопрос – каким образом в условиях данных ограниченных ресурсов будут подтверждаться заявленные требования. На этапе стратегии определяется вид тестирования и каким образом оно будет проводиться.

***12.09***

Принцип присутствия дефекта – если функционал протестирован, и багов не обнаружено6 это не означает что их нет.

Принцип пестицида – способность багов адаптироваться к существующим тестам. (Вместе с модификацией кода, необходимо модифицировать тесты)

Принцип раннего тестирования – стоимость исправления бага, экспоненциально-пропорционально времени его обнаружения.

Модели разработки:

Итерационная модель разработки.

Весь процесс разработки разделяется на этапы(«спринты»), обычно одинаковой продолжительности, до 4х недель.

В рамках спринта реализуются задачи, взятые из «бэклога», относительно их приоритета.

За время спринта производится полный цикл разработки, относительно задач, которые мы реализуем в рамках этого спринта.

Каждый спринт – маленькая завершенная разработка. После релиза спринта, определяется список(«скоуп») задач, которые либо можно, либо необходимо реализовать.

Водопадная модель разработки.

Весь процесс разработки делится на этапы и к следующему этапу можно переходить только в случае, когда предыдущий этап полностью завершен. Если в процессе реализации какого-то этапа, возникла необходимость изменений на предыдущем этапе, в этом случае производится откат на предыдущий этап, он прорабатывается до полного завершения.

Неоспоримым плюсом этой модели является предсказуемость результата.

Минус – увеличенное время-деньги.

Гибкая модель разработки.

В рамках разработки проекта, все этапы выполняются одновременно. При необходимости внесения изменений, изменения вносятся прямо здесь же.

Неоспоримым плюсом является быстрая скорость разработки, меньшие затраты.

Минус – непредсказуемый результат.

Жизненный цикл тестирования:

1. Общее планирование и анализ требований
2. Уточнение критериев приемки (этап, на котором мы выясняем, что на самом деле хочет заказчик)
3. Уточнение стратегии тестирования
4. Разработка тест-кейсов
5. Выполнение тест-кейсов
6. Фиксация найденных дефектов
7. Анализ результатов тестирования
8. Отчётность

Принцип концентрации багов – 80% багов сконцентрировано в 20% приложений.

Основные места концентрации ошибок:

1. платформа Apple.

2. платежные системы.

3. интеграция с любыми сторонними сервисами.

В требованиях необходимо описывать то, что имеет значение для приложения или проекта в целом.

Источники пути выявления требований:

1. Интервью (проводится с диктофоном, запись прослушивается и выписывается)
2. Работа с фокусными группами
3. Анкетирование (необходимо учитывать, что какой-то процент анкетируемых проставит случайные ответы)
4. Семинары и мозговой штурм
5. Наблюдение (очень важно фиксировать результаты наблюдений)
6. Прототипирование
7. Анализ документов
8. Моделирование процессов и взаимодействие
9. Самостоятельное описание

***19.09 Классификация тестирования***

Упрощённая классификация.

По запуску кода на исполнении:

1. Статическое (без запуска кода
2. Динамическое (с запуском)

По доступу к коду и архитектуре:

1. Метод белого ящика (доступ есть)
2. Чёрного (доступа нету)
3. Серого(частично)

По степени автоматизации:

1. Ручное тестирование
2. Автоматизированное тестирование

По уровню детализации тестирования:

1. Модульное (проверяются отдельные небольшие части приложения)
2. Компонентное
3. Интеграционное (проверяется взаимодействие между несколькими частями приложения)
4. Системное (приложение проверяется как единое целое)

По уровню функционального тестирования:

1. Дымовое тестирование (проверка самой важной, ключевой функциональности, неработоспособность которой делает бессмысленной работу приложения)
2. Тестирование критического пути (проверка функциональности, используемой типичным пользователем, в типичной повседневной деятельности)
3. Расширенное тестирование (проверка всей остальной функциональности, заявленной в требованиях)

По принципам работы с приложением:

1. Позитивное тестирование (Взаимодействие с приложением осуществляется по инструкции, без недопустимых действий, некорректных данных и т. п.
2. Негативное тестирование (В работе с приложением выполняются некорректные операции, некорректные данные, оно не предполагает возникновение багов, предполагает корректную обработку неверных данных)

Подробная классификация тестирования (изъятия):

1. Статическое тестирование (документы, графические прототипы (дизайн), код приложения, параметры, параметры среды исполнения приложения, подготовленные тестовые данные.
2. Ручное тестирование
3. Автоматическое(автоматизированное): Плюсы – существенно быстрей, отсутствие человеческого фактора в процессе тестирование, минимизация затрат, способность генерировать тесты неспособные сделать вручную, способность средства автоматизации собирать, сохранять, автоматизировать и предоставлять в удобном виде всю информацию, способность тестов на низкоуровневой системе. Минусы – необходима более высокая квалификация персонала, высокие затраты на сложные средства автоматизации, на разработку и сопровождение автоматизированных тестов, автоматизированные тест-кейсы необходимо постоянно поддерживать в актуальном состоянии, вплоть до написания новых и удаления старых, автоматизация требует более тщательного планирования и управления рисками, увеличения процентов покрытия кода увеличивает риски, средств автоматизации много

После системного тестирования проводятся альфа-, бета-, гамма-тестирования.

Классификация по природе приложения:

1. Тестирования веб-приложений
2. Тестирование мобильных приложений

Классификация по фокусировке на уровне архитектуры приложения:

1. Тестирование уровня представления (взаимодействие приложения со «внешним миром»)
2. Тестирование уровня бизнес-логики (проверка основного набора функций приложения)
3. Тестирование уровня данных
4. Регрессионное тестирование (тестирование, направленное на проверку того факта, что в ранее работоспособной функциональности не появились ошибки, вызванные изменениями приложения или в среде его функционирования)

ДЗ: расширенная версия классификаций тестирования

***26.09 Регресс***

Регресс проводится после каждого спринта. В случае ограниченных ресурсов регресс приложения проводится со следующими приоритетами:

1. Проверяется новый функционал
2. Проверяется критический путь (аналог smoke-test)
3. Проверяется основной путь (наиболее типичное, стандартное использование приложения)
4. Всё остальное

Чек-лист – набор идей по:

1. Тестированию
2. Разработке
3. Планированию
4. Управлению

Чек-лист в прикладном понимании – список того, что должно в приложении быть и работать, в каждом пункте отражается несколько моментов:

1. Порядковый номер
2. Название требования с описанием сути
3. Приоритет требования:
   1. Критический
   2. Важный
   3. Средний
   4. Неважный
4. Возможные предусловия

Тест-кейс и его жизненный цикл.

Тест-кейс – набор входных данных условий выполнения и ожидаемых результатов, разработанный с целью проверки того или иного свойства или поведения программного средства.

На прикладном уровне, тест-кейс – формально описанный и записанный способ проверки того или иного функционала приложения.

Тест-кейсы пишутся на основании требования и чек-листа, тест-кейсы раскрывают каждый пункт чек-листа.

Тест-кейсы проверяющие каждый отдельный пункт чек-листа должны быть собраны в единый документ в формате md.

При реализации нового функционала дополняется чек-лист и пишутся новые тест-кейсы на новый функционал.

Регресс представляет из себя полное выполнение тест-кейсов, написанных на приложение.

Жизненный цикл тест-кейсов:

1. Создан
2. Запланирован
3. Не выполнялся
4. Выполняется
5. Пропущен
6. Провален
7. Пройден успешно
8. Заблокирован
9. Закрыт
10. Требует доработки

Атрибуты тест-кейса:

1. Идентификатор (Название\_Номер требования из чек-листа\_Номер тест-кейса)
2. Приоритет
3. Связанное с тест-кейсом требование из технического задания
4. Модуль или подмодуль приложения
5. Заголовок (суть) тест-кейса
6. Исходные данные необходимые для выполнения тест-кейса
7. Шаги тест-кейса – действия совершаемые тестировщиком при выполнении тест-кейса (только необходимые шаги)
8. Ожидаемый результат по каждому шагу тест-кейса
9. Ссылка на требования

Примеры тест-кейсов на авторизацию:

Тест-кейс 1: успешный вход по логину

При вводе правильных логина и пароля,

Когда я пытаюсь войти в систему,

Я успешно вхожу в систему.

Тест-кейс 2: неверный пароль

При вводе неверного пароля для действительного имени пользователя,

Когда я пытаюсь войти в систему,

Появляется сообщение об ошибке, указывающее на неправильный пароль.

Тест-кейс 3: пустое поле имени пользователя.

При вводе пустого поля имени пользователя,

Когда я пытаюсь войти в систему,

Я должен увидеть сообщение об ошибке, указывающее на то, что поле имени пользователя является обязательным.

Тест-кейс 4: пустое поле пароля.

При вводе пустого поле пароля,

Когда я пытаюсь войти в систему,

Я должен увидеть сообщение об ошибке, указывающее на то, что поле пароля является обязательным.

Тест-кейс 5: успешный вход по почте

При вводе правильных почты и пароля,

Когда я пытаюсь войти в систему,

Я успешно вхожу в систему.

***3.10***

Должны быть написаны тест-кейсы на каждый запрос бекенда, в том числе уже на существующие запросы. Должен быть проверен вариант корректного ответа и все возможные варианты ошибок.

Свойства качественных тест-кейсов:

1. Правильный технический язык, точность и единообразие формулировок (Лаконичность и понятность описания)
2. Использовать безличную форму глагола.
3. Обязательно указывать точные имена и технически верные названия элементов приложения.
4. Не надо объяснять базовые принципы работы с компьютером.
5. Необходимо следовать правилам написания тест-кейсов на проекте.
6. Баланс между специфичностью и общностью.

Минусы излишней специфичности:

1. При повторном выполнение тест-кейса будут выполняться одни и те же действия с одними и теми-же данными
2. Возрастает время написания и доработки тест-кейсов
3. Выполнение тривиальных действий требует дополнительных мыслительных ресурсов

Недостатки излишней общности:

1. Тест-кейс сложен для выполнения
2. Повышен риск халатного отношения к тест-кейсу
3. Тестировщик может неправильно понять т. з.

Баланс между простотой и сложностью - простые вещи должны быть очевидны, сложные вещи должны быть максимально упрощены.

«Показательность» (высокая вероятность обнаружения ошибки)

Последовательность достижения цели.

Отсутствие лишних действий.

Не избыточность по отношению к другим тест-кейсам.

Демонстративность – способность продемонстрировать обнаруженную ошибку очевидным образом.

Прослеживаемость – из тест-кейса должно быть понятно какую часть приложения проверяем.

Возможность повторного использования.

Повторяемость.

Соответствие принятым шаблонам оформления и традициям.

Отчёт о дефектах.

Фактический результат – поведение системы, наблюдаемое в процессе тестирования.

Под ошибкой мы понимаем действие человека приводящие к некорректным результатам.

Дефект – недостаток в компоненте или системе способный привести к ситуации сбоев или отказов.

Сбой – самоустраняющийся или однократный отказ, устраняемый незначительным вмешательством оператора.

При прохождении тест-кейсов багом мы будем называть расхождения или различия между ожидаемым поведением приложения и фактическим поведением.

Атрибуты дефекта (бага):

1. Идентификатор (обычно используется сквозная нумерация (пятизначные идентификаторы – это норма))
2. Краткое описание (название)
3. Подробное описание с указанием ожидаемого результата, фактического результата и пункта из требований
4. Воспроизводимость (Всегда, иногда, плавающие). Баг считается стабильно воспроизводимым если его удалось повторить тестировщику 3 и более раз.
5. Критичность – это приоритет бага относительно функционала приложения.
6. Срочность – приоритет бага относительно бизнеса.
7. Возможность обойти
8. Комментарий
9. Атрибут приложения

Баг считается стабильно воспроизводимым если его удалось повторить тестировщику 3 и более раз.

Шаги по воспроизведению:

* + 1. ***Классы эквивалентности***

Класс эквивалентности - набор данных, обрабатываемых одинаковым образом и приводящих к одинаковому результату.

Граничное условие – значение, находящееся на границе классов эквивалентности.

Граничное значение необходимо проверять на ряду со значениями из класса эквивалентности. Граничные значения являются точками концентрации багов. Тест-кейсов необходимо записать минимально-допустимое количество для того, чтобы их было достаточно для проверки функционала приложения. Диапазон всех значение разбивается на кассы эквивалентности, классы могут быть различные (требования к длине, требование к составу строки). Для каждого множества классов эквивалентности проверяются допустимые значения, несколько недопустимых значений и граничные условия.

Тоже самое делаем для другого множества классов эквивалентности. Тест-кейсы могут содержать в себе различные проверки классов эквивалентности.

Отчет о дефектах и реграссах:

При завершении цикла разработки(спринта), необходиио произвести регресс существующего функционала (желательно полный, хотя бы критического пути) и полный регресс нового функционала. При обнаружении дефектов оформляются баги из расчёта на 1 дефект – 1 баг. Баг является задачей исполнителем которой в общем случае является разработчик, в задаче которого обнаружен дефект. После исправления багов, стабилизации приложения оформляется баг-репорт. В баг-репорте отмечается:

1. Дата проведения регресса
2. Какое приложение тестировали
3. Версия приложения
4. Какой функционал был затронут регрессом
5. Список проведенных тест-кейсов
6. Список не проведенных тест-кейсов с пометкой почему они не были проведены
7. Список багов, которые были найдены (со статусом исправлен/не исправлен).

Не исправленные баги в текущем релизе выносятся в тех док.

Каждый баг репорт оформляется отдельным документом и хранится в репозитории.

Тимлид нарезает задачи, у задачи должен быть номер и описание. Каждая задача реализуется в отдельной ветке. На конкретную задачу тестировщик пишет тест-кейсы.

ДЗ:

Прототип! ТЗ! Чек-листы! Тест-кейсы!

21.11 HTTP

Hypertext transport protocol - протокол передачи информации между клиентом и сервером. HTTP – это шаблон по которому формируется запрос.

Структура HTTP сообщения:

1. Стартовая строка – адрес запроса, тип сообщения.
2. Метод – действие при получении сообщения (чтение, правка, изменение, удаление данных)
3. Заголовки (headers) – прописываются параметры запроса. Обязательные хедеры:
   1. Тип запроса, версия протокола.
   2. Хост – адрес, с которого отправляется запроса
   3. User агент – строка, в которой явным образом прописано приложение и версия приложения, с которого отправляется пользовательский запрос
   4. Используемая кодировка
   5. Используемый язык
   6. Формат ответа
4. Тело запроса (request body) – данные, которые передаются.

Помимо HTTP существуют другие протоколы:

1. FTP – для передачи файлов
2. SSH – для удалённого управления компьютером
3. UDP

HTTP организует передачу данным в зашифрованном виде.

Основные методы HTTP:

1. Get - используется для запроса некоторых данных с сервера. (запрос остаётся в истории браузера и может быть сохранён в закладках, у этого запроса отсутствует тело запроса
2. Post – используется для отправки данных на сервер, для создания или обновления запрашиваемого ресурса, данные указываются в теле. Запрос не сохраняется в истории браузера и не может быть сохранён в закладках.
3. Put – используется для отправки данных на сервер. Отличие от post в том, что put не требует ответа от сервера.
4. Patch – используется для частичного изменения ресурса. Включает в себя
5. Delete – используется для удаления запрашиваемого ресурса.
6. Options – возвращает описание ресурса.

Преимущества HTTP:

1. Расширяемость
2. Подробна документация
3. Распространённость
4. Удобство
5. Стабильность работы

Недостатки:

1. Относительно других протоколов HTTP гораздо медленнее.
2. Отсутствие навигации (не позволяет запросить все доступные ресурсы и их параметры)
3. Проблемы с масштабируемостью

Система HTTP.

Система HTTP требует минимум два участника – один из них - клиент, которые отправляет запросы (браузер, приложение), второй - сервер, который отвечает на запросы клиента. В качестве клиента может выступать сервер. Запросы высылает клиент, сервер не может послать запрос в клиент.

Порядок работы:

1. Формирование URL или переход по введённой ссылке в браузере клиента.
2. Клиент формирует и отправляет запрос.
3. Запрос направляется напрямую на сервер, либо через прокси.
4. Движение запроса регулируется протоколом TCP/iP.
5. Сервер получает запрос, обрабатывает и формирует ответ.
6. Клиент получает ответ и отображает результат.

Изучить порядок действий протокола HTTPS.

Статус коды

1. C1 – информационные
2. 200е – успешные запросы, означают что запрос на доступ к ресурсу выполнен успешно
3. 300е – редирект или перенаправление. Указывают на то, что запрос клиента отправлен на уру отличный от исходного.
4. 400е – ошибка клиента. Ошибка на стороне клиента, может быть связана с неверным форматом, отсутствием доступа
5. 500е – ошибка сервера. Клиент сделал правильный запрос, но сервер не может сгенерировать запрошенный ресурс.

Любому запросу на бекенде в ответе можно прописать любой статус код.

Программа Postman – инструмент для отправки запросов.

У postman есть структура каталога:

1. Коллекция. Внутри коллекции могут быть папки.
2. Параметры запроса. Есть вкладка авторизация, где создаются авторизационные токены, которые в свою очередь прописываются в хедер запроса. Скрипты, которые отработают до момента отправки запроса.
3. Тест. Автотесты которые будут отрабатываться во время этого запроса.
4. Variables. Переменные, которые будут доступны во время запроса.

Тесты и все остальные скрипты пишутся на js. Переменные можно добавлять как локально для како-то коллекции, так и глобально. Переменные коллекции:

1. Initial value – увидят все кто работает с коллекцией
2. Current value – увидите только вы

Глобальные переменные – environment.

В JS нет ассертов – специализированная команда, которая при сравнении бросает exception. Ответ может быть различно

Данные для postman могут храниться в отдельно подгружаемых файлах.

Коды ошибок:

1. Код 100
2. 101
3. 102 – процессинг. Сервер получил запрос и обрабатывает его.
4. 103 – early things. Ранние подсказки, разрешение пользовательскому агенту начать загрузку пока сервер готовит ответ.
5. 200 – запрос успешно обработан
6. 201 – created. Сервер подтвердил создание ресурса.
7. Код 202 – accepted. Запрос клиента был получен, но сервер всё ещё обработывает его.
8. Код 203 – non authored information.
9. Код 204 – no content, сервер обработал запрос, но нифего не ответил
10. 205 reset content. Сбросить содержимое, клиент должен обновить документ который прислал запрос
11. 206 – partial content. Частичное содержимое – клиент присылает заголовок диапазона чтобы выполнить загрузку отдельно в несколько потоков.
12. 300 – multiple choice – запрос имеет более чем 1 из возможных ответов, либо юсер агент либо пользователь выбирает один из ответов
13. 301 moved permanently – искомый ресурс перемещён по другому адресу, все пользователи ресурса перенаправляются
14. Код 302 – found – ресурс временно перемещён на другой URL
15. Код 303 – see other – сообщает клиенту что сервер перенаправляет его не на запрашиваемый ресурс, а на другой
16. 304 – используется для кэширования, означает что запрошенный ресурс не был изменен после последнего посещения
17. Код 305 – use proxy – клиент может получить доступ только через прокси указанным в ответе
18. 307 – temolary redirect – искомый ресурс был временно перенесен на другой url(нелья изменять используемый http метод)
19. 308 Permanent redirect –
20. 400 – bad request – ошибка в синтаксисе
21. 401 – unauthoruzed - неавторизован
22. 402 – payment req – занеси деньгу
23. 403 – forbidden – нет прав
24. 404 – not found – сервер не может найти запрашиваемый ресурс
25. 405 – method not allowed – сервер получил запрос, распознал его, но конкретный метод запроса не разрешен
26. 406 – not acceptable - сервер не поддерживает запрос клиента по этому протоколу
27. 407 – proxy authefication reauire
28. 410 – gone - запрашиваемый контент навсегда удален
29. 413 – precondition failed
30. 414 – request url too long
31. 417 – expectation failed – ожидаение полученное из заголовка запроса expect не может быть выполнено сервером
32. 418 – im teapot – я чайник
33. 500 – eternal server error – внутренняя ошибка сервра, сервер стоклнуля с истуацией
34. 501. – not implemented – не реализовано
35. 502 – bad gateway
36. 503 – сервер недоступен
37. 504 – gateway timeout – сервер действует как шлюз и не может получить ответ
38. 505 – http version is not supported
39. 511 – network aunthification require - требуется сетевая аунтефикация