***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

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:

***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.