Препод – Евгений Александрович

Введение в тестирование ПО

Тестирование ПО – процесс анализа ПО и её документации с целью выявления дефектов и повышения качества продукта.

Тестирование ПО является процессом в силу того факта, что оно выполняется большим кол-вом людей на протяжении длительного периода времени.

Тестировать нужно:

1. Программы
2. Код программ
3. Прототип ПО
4. Документация
   1. Требования к ПО
   2. Функциональные спецификации
   3. Архитектуру
   4. План проекта
   5. Тестовые случаи сценария
5. Сопроводительную документацию
6. Интерактивную помощь
7. Рук-ва по установке и использованию

Дефект (баг, фича) – Не соответствие фактического и ожидаемого результата

Ожидаемый результат – поведение ПО в ответ на наши действия

Тест-кейс – набор входных данных, условий выполнения и ожидаемых рез-тов, разработанный с целью проверки того и иного свойства поведения ПО. (сценарии для теста)

Тест-план – Часть проектной документации, описывающая пи регламентирующая процесс тестирования.

Билд – промежуточная версия ПО; Релиз – итоговая версия ПО;

Статическое тестирование – Процесс анализа самой разработки ПО, т.е. без запуска ПО

Динамическое тестирование – тестовая деятельность, предусматривающая эксплуатацию продукта.

Методы тестирования:

Метод белого ящика – тест без запуска. Тестировщик имеет доступ и исходному коду, тесты основаны на его знании кода.

Метод белого ящика используется тогда, когда ПО не собрано воедино и его нельзя пускануть.

Метод чёрного ящика – он заключается в том, что таксировщик имеет доступ к ПО только через интерфейсы, которые будут иметь заказчик или пользователь.

Тестирования ведётся с исп. спецификаций или иных док-тов, описывающих требования к системе на основе применения пользовательского интерфейса для ввода входных и получении выходных данных

Цель метода – проверить работу всех функция приложения на соответствие функциональным требованиям.

Метод серого ящика – совокупность подходов белого и чёрного ящика.

Этот метод используется при тестировании веб-приложения, когда татуировщик знает принципе функционирования технологий, на которых оно построены, но не может видеть код самого приложения

Уровни тестирования:

Компонентное тестирование – тестирование отдельного модуля (в т.ч. класс и метод)

Интеграционное тестирование – проверка взаимодействия модулей

Системное тестирование - полная проверка ПО

Функциональное тестирование - Процесс проверки ПО, сконцентрированный на анализе соответствия ПО требования и спецификацию. Функциональное тестирование, бывает ручным и автоматизированным

Его цели:

* + 1. Обнаружить дефекты
    2. Определить степень соответствия ПО ожиданиям заказчика
    3. Принять решение о передачи ПО заказчику

Приёмочный тест - самый первый и быстрый уровень. Если тест не проходит – тест прекращается.

Тест критического пути – тестер имитирует пользователя

Расширенный тест – Тест всего функционала продукта

Качество – показатель степени соответствия продукта предъявляемым к нему требованиям. Качество продукта определяется качество процесса его разработки.

Регрессионное тестирования – Проверка на то, что новый код не сломал старый

Тестирование нового функционала – проверка новых фич.

Конфигурационное тестирование – проверка того, как приложение работает с различными настройками и оборудованием.

Тестирование совместимости - Проверка того, как приложения взаимодействуют с ОС и другими приложениями.

Тестирование удобства использования – Проверка того, насколько юзеру удобно юзать ПО

Тестирования интернационализации – проверка возможности и качества перевода

Тестирование локализации –проверка качества перевода продукта на конкретный язык

Исследовательское тестирование – Тест, основанный на профессиональной интуиции тестировщика, которая может предсказать опасные и малоисследованные способы.

Позитивное тестирование – проверка в тепличных условиях

Негативное тестирование – проверка с ПрЫкОлАмИ (в плохих условиях)

Основные типы требований:

1. Функциональные (что система должна делать);
2. Нефункциональные (с соблюдением каких требований она должна это делать (Время, скорость отклика)).

Уровни требований:

1. Бизнес требования;
2. Пользовательские требования;
3. Функциональные требования.

Хорошо проработанные требования позволяют:

1. Выработать общее понимание между разрабом и заказчиком;
2. Определить рамки проекта;
3. Более точно определить финансовые и временные хар-ки проекта;
4. Обезопасить заказчика от риска получить продукт, который ему не нужен;
5. Обезопасить заказчика от потери денег посредством «неконтролируемого; размытия границ».

Пути выявления требований:

1. Интервью;
2. Наблюдение;
3. Самостоятельное описание;
4. Семинары;
5. Прототипирование.

Стандарт IEEE 830-1998 – описывает развёрнутое описание требований, которые может быть оптимизировано для нужд конкретной организации

В описание требований рекомендуется включать:

1. Введение
   1. Назначение док-та;
   2. Поддерживаемые соглашение;
   3. Предполагаемая аудитория и рекомендации по последовательности работы с док-том для каждого класса читателей;
   4. Границы проекта и ссылка на «концепцию»;
   5. Ссылки;
2. Общее описание
   1. Общий взгляд на продукт (что есть продукт);
   2. Особенности продукта;
   3. Класса и хар-ки пользователей;
   4. Операционная среда (где работает);
   5. Ограничение проектирования и реализации (На каких языках можно и нельзя это писать, что не так с ОС, минимальные системные требования, цвета интерфейса и т.д.);
   6. Документация для пользователей;
   7. Предложения и зависимости;
3. Функции системы
   1. Наименование i-й функции системы;
   2. Описание и приоритеты этой функции;
   3. Последовательности «воздействия-реакции» для данной функции;
   4. Функциональные требования этой функции;
4. Требования к интерфейсу
   1. Ссылки на стандарты UI;
   2. Стандарты шрифтов, значков, кнопок и т.д.;
   3. Конфигурация экрана или ограничение разрешения;
   4. Стандартные кнопки, функции или ссылки одинаковые для всех экранов;
   5. Хоткеи;
   6. Стандарты отображения сообщений;
   7. Стандарты конфигурации для упрощения локализации ПО;
   8. Спец. Возможности для людей с проблемами со зрением;
   9. Интерфейсы оборудования;
   10. Интерфейсы ПО;
   11. Интерфейсы передачи информации;
5. Другие нефункциональные требования
   1. Требования к производительности;
6. Приложения
   1. Приложение А. Словарь терминов (глоссарий);
   2. Приложение Б. Модели анализа (Все модели, построенные в процессе анализа требований);
   3. Приложение В. Список вопросов (не решённые проблемы).

Каждое требование должно быть:

1. Завершённым;
2. Непротиворечивым;
3. Корректным;
4. Недвусмысленным;
5. Проверяемым;
6. Модифицируемым;
7. Прослеживаемым;
8. Проранжированным по важности, стабильности и срочности.

Проблемы незавершённости:

Проблема незавершённости – когда мы знаем, что должны получить, но хз как и с какими характеристиками мы должны это делать.

Для того чтобы избежать таких приколов, нужно спрашивать общие вопросы, типо «как должно работать, на чём, где, зачем». Их преимущество:

1. Они универсальные;
2. Они не навязывают решение;
3. Они не загоняют заказчика в ситуации, когда ему нужно выбивать одну ситуацию из двух.

Ещё одна проблема незавершённости – TBD (To be Defined, или «Будет определено»)

Проблемы противоречивости:

Проблемы противоречивости – это тогда, когда есть противоречие между двумя требованиями, или внутри одного требования.

Для их устранения, стоит просто прочитать требования хотя бы раз, а также нужно определить, что требование во всех частях называется одинаково.

А также, нужно проверить что требование «А» не противоречило:

1. Требованиям про «А»;
2. Требованиям более общим покрывающими «А»;
3. Требованиям, которые связаны с «А»;
4. Требования, про что-то другое, но аналогичные требованию «А»;
5. Как следуя требованию «А», мы можем сломать другие требование.

Проблемы некорректности:

Ошибки некорректности могут быть вызваны:

1. Опечатками;
2. Остатками устаревших требований;
3. Наличием «озолочением» - догорим, но useless функционалом;
4. Наличием технически невыполнимых требований;
5. Наличием неаргументированных требований.

Проблемы двусмысленности;

«Если что-то можно понять двумя способами, это обязательно произойдёт»

За сим, нужно писать требования так, чтобы исключить неверность их понимания.

Типо «Приложение должно уметь принимать большие файлы». Тут нужно уточнить насколько большие, ибо для одного 1мб это много, а для другого 1гб эта мало.

Проблема непроверяемости:

Проблема непроверяемости – это тогда, когда нельзя проверить требование.

Например: «В приложении должно быть 0 ошибок» или «Приложение должно поддерживать все версии всех ОС».  
Даже чисто теоретически, проверить на это нельзя.

Такое ваще не должно быть, ибо тестерам нужно чтобы было что тестировать.

Проблемы непроранжированности:

Проблема непроранжированности – это тогда, когда нету ранжирования требований, и хз чему уделять внимание.

Хорошие требования делятся по:

1. Важности;
2. Приоритетности выполнения;
3. Стабильности (шанс на изменение требования).

**Важно про требования:**

1. Набор требований немодифицируем (из-за того, что в случае малейших изменений, уходит оч много времени на их склейку и вылавливание всех противоречий);
2. Набор должен быть проранжирован (для того, чтобы запилить для начала важное, а фичи допиливать позже);
3. Набор должен быть прослеживаемый (должны быть ссылки на все прыколы от заказчика, чтобы не забывать, что уже запилено или нужно будет допилить

Одна из техник работы с требованиями – **Взаимный пересмотр**

Взаимный пересмотр делится на:

1. Неформальный перепросмотр (тестеры просто свапаются листочками с требованиями, и ищут ошибки);
2. Технический перепросмотр (то же самое, только официально. Что требует подготовки, специалистов и прочее);
3. Формальная инспекция (херня для оч крупных проектов, и в случае больших проектов. Описывается специальными стандартами, требует соблюдения широкого спектра правил и протоколирования результатов).

Способы работы с требованиями:

1. Закидывать вопросами заказчика, коллег. Тем самым, уточняя как представление как и что должно работать;
2. Создавать тест-кейсы – когда мы видим требование, нужно спросить себя «Как это тестировать?». Если хз как – с требованиями проблемы;
3. Рисовать рисунки, схемы и диаграммы.

**Планирование тестовых испытаний**

ЖЦ ПО в контексте тестирования удобнее всего:

1. Начало (Сбор пожеланий заказчика и его требований);
2. Уточнение (Проводятся юниттесты и интеграционные тесты. Прогеры работают);
3. Разработка (Разрабы потеют и допиливают программу);
4. Передача заказчику (Приёмочное тестирование и разворачивание проги у заказчика, проводится пиар-компания);

Чтобы достигнуть успеха, нужно сконцентрироваться на 3-х задачах:

1. Разработка тестов
   1. Разработка методологии и плана тестирования;
   2. Участие в установлении стандарта качества;
   3. Разработка спецификаций тестов;
2. Разработка и выполнение тестов
   1. Разработка и поддержка ручных и автоматизированных тестов;
   2. Выполнение тестов;
   3. Управление билдами (оценка состояния проекта);
3. Отчётность о тестах
   1. Доведение до сведения проектной группы информации о качестве продукта;
   2. Мониторинг ошибок с целью их устранения;

Планирование тестов:

Планирование тестов – команда тестровщиков разрабатывает планы и методики тестирования и таким образом формирует тестовую стратегию, используемую в проекте. План тестирования включает: описание типов тестов, тестируемые составляющие и информацию о требуемых ресурсов.

Планирование - е процесс организации и установления процесса тестирования его описания и согласование его со всеми ключевыми людьми проекта.

Тестовый план – документ, являющийся частью проектной документации, и описывающий что, когда, кем, и как будет тестироваться. Он описывает процесс тестирования конкретного продукта в конкретном проекте.

Ответственным разработку тест-плана – тимлид.

Что нужно сделать на стадии тестирования:

* Понять, что за продукт будут тестировать, как он работает, для чего он нужен;
* Понять, как продукт будет использоваться;
* Разобраться с требованиями;
* Разобраться что будет тестироваться и как;
* Решить какие методы и техники тестирования будут юзаться;
* Определить критерии качества продукта;
* Определить риски (что может сломаться во время разработки. Напр. Болезнь тимлид) и предупредить их возникновение;
* Выводы по всему вышеперечисленному записать в тест-план;
* Озаботиться на тестовом оборудовании.

Артефакты, создаваемые на стадии планирования:

1. Тестовый план;
2. Матрица конфигураций (Все версии билдов и чё там поменялось);
3. Запрос на выделение тестового оборудования.

Риск - сочетание вероятности наступления события и последствие события;

Типо, шанс падения с высотки и последствия падения – риск.

Перечень работ – Перечень фич, которые будут подвергаться тестированию.  
Здесь же, есть перечень фич, которые не будут подвергаться тестированию, в связи с тем, что они реализуются не нами, а другими разрабами, и мы надеемся на них.

Критерии качества и оценка качества процесса – здесь отображается перечень критериев качества, на основании которых будет приниматься заключение об уровне качества продукта и возможности передачи продукта заказчику. Критерии качества относятся к качеству продукта. В текстовых планах могут быть записаны и критерии качества самого процесса тестирования, с целью последующий оценки качества процесс.  
Это необходимо чтобы в случае необходимости существовала возможность оценить, насколько грамотно был построен процесс тестирования, были ли какие-либо проблемы, и , если были, разобраться почему и как их предотвратить их в будущем.

Оценка рисков – Описывается риск, его вероятность и степень его влияния на проект.

Документация и письма – тут размещается перечень артефактов (тест план, тестовые сценарии, тестовые автоматизированные скрипты, дефект-репорты, отчёты о результатах тестирования).

Тестовая стратегия – Описывается стратегия тестирования, методы и типы тестов, каким образом будет выполняться тестирования. Содержание зависит от разраба и заказчика и конкретного проекта.

Артефакты - то что по факту остаётся после тестирования (итоги тестов, ошибки, которые были после них выявлены и т.д.)

Ресурсы - Перечень ключевых людей на проекте. Список тестировщиков с их ролями на проекте и их зоны ответственности.  
Помимо человеческих, перечисляются hardware + software resources.

Расписание и ключевые точки – тут описывается график тестирования в согласовании с графиком выпуска билдов и проектным планом, который разрабатывается менеджером проекта. Сюда же включается основные даты (напр. Даты окончания промежуточных фаз работы над проектом)

**Критерии хорошего тест-плана:**

1. **Полным, корректным и недвусмысленным;**
2. **Должны быть определены цели тестирования;**
3. **Должны быть определены – тестовый подход, стратегия, методы, виды и типы тестирования;**
4. **Запланированный подходи должен быть выполним;**
5. **Должно быть определено тестовое оборудование, окружение, ПО;**
6. **Должны быть установлены адекватные критерии качества;**
7. **Должны быть определены критерии прохождения приёмочного теста и условия прекращения тестирования;**
8. **Должны быть определены все артефакты, подлежащие сдаче;**
9. **Должны быть перечислены тестовые рес-сы;**
10. **Должен быть определён график тестирования;**
11. **Тест план должные быть принятому в компании шаблону (если в проекте не решено иначе).**

**Преимущество хорошего тест-плана:**

1. **Позволяет грамотно организовать действия;**
2. **Позволяет управлять процессом более эффективно;**
3. **Позволяет узнать минимальный уровень тестов;**
4. **Позволяет согласовать график проведения тестов с заказчиком.**

**(Исправить буллщит с подчёркиванием)**

**Тестировщик**

**(Я забыл написать тему)**

Важные правила описания ошибок:

1. Как можно подробнее описать что и в какой последовательности было сделано;
2. Описать симптомы ошибок;
3. Прикладывать логи, файлы, конфигурации (хар-ки пк) и скриншоты;

Описание дефекта должно включать:

1. Предусловия;
2. Навигация к проблемной функции;
3. Перечень шагов по воспроизведению;
4. Описание рез-та (дефекта);
5. При необходимости, указать правильное поведение системы, привести пример из документации.

*При описании дефекта, важно докопаться до сути дефекта*

Необходимость документирования:

1. Зафиксированная проблема будет исправлена программистом с большей вероятностью;
2. В электронном описании содержатся и подробности воспроизведения проблемы;
3. Отчёт нужен для проверки;
4. Другой таксировщик на проекте может посмотреть предыдущие отчёты;
5. Программист сопровождающий прогу, может узнать является ли тот костыль на который он смотрит является вынужденной мерой для исправления ошибки;
6. Запись сохраняется для дальнейшего рассмотрения руководством.

Уровень ошибок:

1. Тривиальные
2. Мажорные
3. Критические
4. Блокер

Неграмотное написание отчёта об err может привести:

1. Дефект может быть исправлен неверно;
2. По дефекту могут возникнуть доп. Вопросы, что приведёт к трате времени;
3. При регрессивном тестировании возможно нельзя будет воспроизвести эффект;
4. Разраб возможно не сможет понять в чём заключается дефект;
5. Разраб возможно не сможет воспроизвести дефект.