**Преподаватель: Скиндер Оксана Геннадьевна**

**Лекция 1**

Тестирование программного обеспечения:

* Поиск ошибок
* Проверка соответствия ПО требования и здравому смыслу
* Оценка работоспособности ПО
* Способ контролировать качество ПО

Причины ошибок ПО:

* Человеческий фактор
* Проблемы в описании требований к программному обеспечению
* Недостаток времени
* Недостаточно продуманная архитектура приложения
* Недостаточное знание бизнеса
* Нехватка профессиональных навыков и опыта
* Изменения “в последнюю минуту”

История тестирования программного обеспечения:

Первые программные системы разрабатывались в рамках программ научных исследований или программ для нужд министерств обороны.

Тестирование таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых данных, полученных результатов.

Тестирование выделялось в отдельных процесс, который начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

В 1960-ч много внимания уделялось “исчерпывающему” тестированию, которое должно проводиться с использование всех путей в коде или всех возможных входных данных.

Однако это невозможно:

#1 количество возможных входных данные очень велико

#2 существует множество путей

#3 сложно найти проблемы в архитектуре и спецификациях

Итог: “исчерпывающее» тестирование было отклонено и признано теоретически невозможным.

В начале 1970-ч годов тестирование программного обеспечения обозначалось как “процесс, направленный на демонстрацию корректности продукта” или как “деятельность по подтверждению правильности работы программного обеспечения”

Впоследствии этот метод тестирования был признан неэффективным.

Во второй половине 1970-ч тестирование представлялось как выполнение программы с намерением найти ошибки, а не доказать, что она работает

Успешный тест – это тест, который обнаруживает ранее неизвестные проблемы.

В 1980-е годы тестирование расширилось таким понятием, как предупреждение дефектов.

Стали высказываться мысли, что необходима методология тестирования в частности, что тестирования должно выключать проверки на всем протяжении цикла разработки, и это должен быть управляемый процесс.

В золе тестирования надо проверить не только собранную программу, но и требования, код, архитектуру, сами тесты.

Для разработчиков обеспечение качества это: Unit тестирование, code review, попарное программирования, а тестировщики обеспечивают качество за счет тестирования требований.

В начале 1990-ч годов в понятие “тестирование” стали включать планирование, проектирование, создание, поддержку и выполнение тестов и тестовых окружений, и это означало переход от тестирования к обеспечению качества.

Начинают появляться различные программные инструменты для поддержки процесса тестирования: более продвинутые среды для автоматизации с возможностью создания скриптов и генерации отчетов, системы управления тестами, ПО для проведения нагрузочного тестирования.

Тестирование/обеспечение качества

Тестирование: проверяем рабочий код на наличие ошибок, это контроль качества.

Обеспечения качества: это ряд мер, в нем участвуют все.

Тестирование – quality control.

Обеспечение качества – quality assurance.

Тестированием занимаются:

* Заказчики (ручное тестирование (приёмочное тестирование))
* Разработчики (автоматизированное тестирование (модульное тестирование))
* Тестировщики (ручное, автоматизированное, нагрузочное тестирования и тестирование безопасности)
* Бизнес аналитики (ручное тестирование (тестирование требований))
* Команда технической поддержки продукта (ручное тестирование (регрессионное тестирование))
* Дизайнеры (ручное тестирование (тестирование пользовательского интерфейса))
* Конечные пользователи (ручное тестирование (beta-тестирование))

ISO (International Organization for Standardization) (ISO/IEC 25010: 2011)

Качество системы – это степень удовлетворения системой заявленных и подразумеваемых потребностей различных заинтересованных сторон, которая позволяет, таким образом, оценить достоинства.

Эти заявленные и подразумеваемые потребности представлены в международных стандартах серии SQuaRE посредством моделей качества, которые представляют качество продукта в виде разбивки на классы характеристик, которые в отдельных случаях далее разделяются на под характеристики.

Все, кто взаимодействует с приложением либо со стороны создания, либо со стороны использования является заинтересованной стороной.

Модели качества:

* Модель качества при использовании
* Модель качества продукта
* Модель качества данных ISO/IEC 25012:2008

(1,2 в 25010: 2011)

Модель качества при использовании:

* Эффективность(результативность) (на сколько эффективно конечный пользователь достигает свой цели, при использовании приложения)
* Производительность (на сколько быстро система выполняет действия)
* Удовлетворенность (полноценность, доверие, удовольствие, комфорт) (Полноценность – в какой мере удовлетворяет пользователя, Доверие – пр. Http/ Https, Комфорт связан с удобством использования).
* Свобода от риска (смягчение отрицательных последствий:
* Экономического риска
* Риска для здоровья и безопасности
* Экологического риска
* Покрытие контекста (полнота контекста, гибкость) (Полнота контекста – характеристика, кот означает на сколько полноценно приложение выполняет свои функции в условиях: маленького экрана, низкой сетевой пропускной способности, малого количество доступной памяти, малого заряда батареи). (Гибкость – характеристика, кот означает на сколько приложение адаптируется для различных групп пользователей, а именно: пользователи с плохим зрением, пользователи с нарушениями мелкой моторики, пользователи различных культурных принадлежностей, пользователи с различным опытом взаимодействия с данным ПО).

**Лекция 2**

Модель качества продукта:

* Функциональная пригодность (Это степень, кот продукт или система обеспечивают выполнение функции в соответствии с заявленными или подразумеваемыми потребностями):
* Функциональная полнота;
* Функциональная корректность;
* Функциональная целесообразность.
* Уровень производительности (Это производительность относительно суммы использованных при определенных условиях ресурсов):
* Временные характеристики;
* Использование ресурсов;
* Потенциальные возможности.
* Совместимость (Это способность продукта или системы, или компонента обмениваться информацией с другими продуктами, системами или компонентами, а также выполнять требования функций при совместном использовании одних и тех же ресурсов, аппаратных средств или программной среды):
* Сосуществование (Способность продукта совместно функционировать с другими независимыми продуктами в общей среде с разделением общих ресурсов и без отрицательного влияния другого продукта).
* Интероперабельность (Это способность двух и более систем, продуктов или компонентов обмениваться информацией и использовать такую информацию).
* Удобство использования (Это степень, которой продукт или система могут быть использованы определенными пользователями для достижения конкретных целей с эффективностью, результативностью и удовлетворенностью в заданном контексте использования):
* Определимость пригодности (Это возможность пользователей понять подходит ли продукт или система для их потребностей)
* Изучаемость
* Управляемость
* Защищенность от ошибки пользователя
* Эстетика пользовательского интерфейса
* Доступность (Это возможность использования продукта или системы широким кругом людей с разными возможностями)
* Надежность (Это степень выполнение системой, продуктом или компонентом определенных функций при указанных условиях в течении установленного времени выполнения):
* Завершенность (Это степень соответствия системы, продукта или компонента требованиям надежности)
* Готовность (Это степень работоспособности и готовности системы, продукта или компонента. Готовность определяется сочетанием готовности, которая определяет частоту отказов, отказоустойчивости и восстанавливаемости, которая определяет продолжительность времени бездействия после каждого отказа)
* Отказоустойчивость
* Восстанавливаемость
* Защищенность (Это степень защищенности информации и данных, обеспечиваемое продуктом, путем ограничения доступа людей в соответствии типом и уровнем авторизации):
* Конфиденциальность
* Целостность (Это степень предотвращения системой, продуктом или компонентом несанкционированного доступа и модификации компьютерных программ)
* Неподдельность
* Отслеживаемость (Это степень, для которой действия объектов могут прослежены однозначно)
* Подлинность (Это степень достоверности тождественности объекта или ресурса, требуемому объекту или ресурсу)
* Сопровождаемость (Это результативность и эффективность с которой продукты или компоненты могут быть модифицированными специалистами по обслуживанию)
* Модульность (Это степень представления системы или программы в виде отдельных блоков, чтобы изменения в одном модуле минимально затрагивали изменения в другом модуле)
* Возможность многократного использования (Это степень, в которой актив может использоваться в других системах или в создании других актив)
* Анализируемость (Это степень простоты оценки влияния изменений в одной части программы на выявления недостатков и причин отказов)
* Модифицируемость
* Тестируемость
* Переносимость (Это степень простоты эффективного и рационального переноса системы из одной среды в другую)
* Адаптируемость (Это степень эффективной и рационально адаптации к совершенствованию аппаратных средств, программного обеспечения)
* Устанавливаемость
* Взаимозаменяемость (Это способность продукта заменить другой конкретный программный продукт, для достижения тех же целей в тех же условиях)

Change request (Запрос на изменение функционала или добавление фичи)

Functional specification (документ с описанием требований ко внешнему виду и работе приложения)

Requirement (требование к приложению, описание юзер-интрефейса)

Feature freeze (Период, когда новые фичи не разрабатываются, только баг фиксы)

Code freeze (Нельзя ничего комитить, запрет на изменение кода, запрет на изменение багов)

Verification (процесс тестирования приложения на соответствие предъявленным требованиям)

Validation (проверка приложения на соответствие бизнес-целям проводится как правило перед релизом)

Лк3(from lab)

Структура описания бага:

* ID (идентификационный номер)
* Title (название)
* Description (описание)
* Component (компонент)
* Priority (приоритет)
* Assignee (ответственный)

Требования к названию дефектов:

* Краткость
* Информативность (Стоит на 1 месте)
* Точная идентификация проблемы

Способ написания названия

**Описание бага должно отвечать на 3 вопроса:**

* Where: где случился баг?
* What: Что именно происходит с приложением?
* When: При каких условиях/действиях происходит баг?

**Description:**

* Preconditions (подготовка, предусловия)
* Steps to reproduce (шаги воспроизведения)(должны быть пронумерованы)
* Actual result (актуальный результат/проблема)(первым актуальный потом ожидаемый)
* Expected result (ожидаемый результат)
* Environment (окружение)
* Notes (комментарии)

(Начало всегда с открытия приложения)

**Expected result** (Ожидаемый результат)

Рекомендации:

* Обоснование (ссылка на конкретный пункт спецификации)
* Выводы из текста спеки
* Исправленный вариант текста с ошибкой
* Безличные предложения с использованием модального глагола should
* Порядок: сначала Actual Result, потом Expected Result

**Attachment** – любой прикрепленный к дефекту файл, облегчающий его понимание.

Скриншот должен содержать след элементы:

* Сама ошибка
* Выделение прямоугольником места ошибки
* Стрелка к прямоугольнику
* Описание ошибки с Actual или Expected result

Правила оформления скриншотов:

* На скриншоте должна быть вся страница, включая Page Title, Address Bar Status Bar
* В браузере не должны быть открыты личные табы
* Область UI бага должна быть выделена красным квадратом или кругом
* Task Bar Windows с открытым окошками должен вырезаться

**Component** (компонент)

Feature, модуль приложения, в котором был найден баг

**Assignee** (ответственный)

Человек, который должен заняться багом после того, как он будет занесен в систему.

**Priority** (приоритет)

Порядок, в котором дефект должен быть исправлен

Можно выделить пять основных классов:

* Blocker – баг блокирует тестирование фичи.
* Critical – баг, когда не выполняется существенная для фичи операция и нет обходных путей, чтобы эту операцию выполнить.
* Major – баг, когда не выполняется (или выполняется неправильно) операция, но есть обходной пусть, чтобы выполнить эту операцию.
* Minor – несущественный функциональный баг, который не влияет на выполнение основных операций фичи.
* Trivial – UI дефекты в приложении.

**Лекция 3. Процесс тестирования ПО**

1. Планирование и управление
2. Анализ и проектирование
3. Внедрение и реализация
4. Оценка критериев выхода и создание отчетов
5. Действия по завершению тестов(Опциональный)

**Планирование тестирования** – это действия, направленные на определение целей тестирования и описание задач тестирования для достижения этих целей, и миссии.

**Управление тестированием** - это постоянное сопоставление текущего положения дел с планом и отчетность о состоянии дел, включая отклонения от плана.

Планирование тестирование:

* Анализ требований;
* Определение целей тестирования;
* Определение общего подхода к тестированию (уровни тестирования, виды тестирования, критерия входа);
* Интегрирование с разработкой ПО (требования, архитектура, дизайн, разработка, тестирование, релиз);
* Решение, какие роли нужны для выполнения тестирования. Когда и как проводить тестирование и как оценивать результаты.

**Критерий входа** – это критерий, по которому мы определяем, когда начинать тестирование, а когда не начинать.

Роли:

* Тест дизайнер, который пишет документацию;
* Тестировщик – который проводит тесты;
* Автоматизатор;
* Аналитик производительности;
* Тестировщик безопасности;

Планирование тестирования:

* Составление графика тестирования;
* Определение шаблонов для тестовой документации;
* Выбор метрик для мониторинга и контроля подготовки и проведения тестирования, исправления дефектов проблем и рисков.

Критерии входа в тестирование (когда нужно начинать тестирование):

* Готовность и доступность тестового окружения;
* Готовность средства тестирования в окружении;
* Доступность тестируемого кода;
* Доступность тестовых данных.

**Анализ и проектирование тестов** – это деятельность, во время которой общие цели тестирования материализуются в тестовые условия и тестовые сценарии.

* Рецензирование базиса тестирования
* Функциональные и нефункциональные требования
* Архитектура
* Дизайн
* Технические требования к интерфейсу
* Оценка тестируемости базиса тестирования и объектов тестирования
* Идентификация и расстановка приоритетов тестирования;
* Выявление необходимых данных для поддержки тестовых условий и тестовых сценариев;
* Проектирование и установка тестового окружения и выявление необходимой инфраструктуры и инструментов;
* Создание двунаправленной трассируемости между тестовым базисом и тестовым сценарием;

**Внедрение и реализация тестов** – это деятельность, где процедуры тестирования или автоматизированные сценарии задаются последовательность тестовых сценариев, а также собирается любая информация, необходимая для выполнения тестов, разворачивается окружающая среда, и запускаются тесты.

* Завершение, реализация и расстановка приоритетов тестовых сценариев (включая проектирование тестовых данны);
* Написание автоматизированных сценариев тестирования;
* Проверка правильности настройки тестового окружения;
* Проверка и обновление двунаправленной трассируемости между тестовым базисом и тестовым сценарием;
* Выполнение процедур тестирования либо вручную, либо используя инструменты выполнения тестов, согласно заданному плану.
* Регистрация результатов выполнения тестов;
* Сравнение фактических и ожидаемых результатов;
* Оформление отчетов об ошибках и занесение их баг-трекинговую систему;
* Повторное тестирование областей, где были исправлены ошибки и областей, где могут появится новые ошибки после исправления уже известных ошибок (регрессионное тестирование).

Критерии выхода и отчетность – это деятельность, где выполнение тестов оценивается согласно определенным целям. Она должна быть выполнена для каждого уровня тестирования.

* Сверка протокола тестирования в сравнении с критериями выхода определёнными в плане тестирования;
* Анализ необходимости использования дополнительных тестов или изменения критериев выхода;
* Написание итогового отчета о тестировании для заинтересованных лиц.

Критерии выхода – когда нужно прекращать тестирование

* Степень покрытия кода, функциональности или рисков тестами
* Оценку плотности дефектов или измерение надежности
* Стоимость
* Остаточные риски (неисправленные дефекты или недостаток тестового покрытия какой-либо области)
* План, основанный на времени выхода ПО на рынок.

**Действия по завершению тестов** – это сбор данных о завершенных испытаниях для объединения опыта, тестового обеспечения, фактов и цифр. Проводится после релиза.

* Проверка, что запланированные результаты достигнуты
* Завершение и архивирование тестового обеспечения, тестового

окружения и инфраструктуры тестирования для последующего использования

* Передача тестового обеспечения организации сопровождения
* Анализ полученных уроков для повышения зрелости процесса тестирования.

Процесс тестирования:

* Планирование и управление
* Анализ и проектирование
* Внедрение и реализация
* Оценка критериев выхода и создание отчетов
* Действия по завершению тестов

Процесс обеспечения качества:

* Тестирование требований
* Реализация unit-тестов
* Реализация интеграционных тестов
* Code review
* Интеграция фичи в систему
* Alpha тестирование
* Отладка
* Beta тестирование

Уровни тестирования ПО:

* Компонентное тестирование (new feature testing vs unit test);
* Интеграционное тестирование;
* Системное тестирование;
* Приемочное тестирование;

Верификация:

* Компонентное
* Интеграционное
* Системное

Валидация:

* Приемочное

**Лекция 4. Типология тестирования (методы тестирования)**

Метод “Белого ящика” – тестирование исходного кода ПО.

Метод “Черного ящика” – тестирование ПО через интерфейс (без доступа к исходному коду)

Метод “Серого ящика” – белый + черный ящики.

Типы тестирования – группы активностей тестирования, направленные на проверку работоспособности системы (или части системы), где за основу принимаются различные цели и причины для тестирования.

Возможные цели:

* Функция, выполняемая программой;
* Нефункциональная характеристика качества (надежность или удобство использования);
* Структура или архитектура программы, или системы;

Типы тестирования:

* Функциональное тестирование;
* Нефункциональное тестирование;
* Структурное тестирование;
* Тестирование изменений.

**Функциональное** – тестирование, которое разрабатывается на основе функций и возможностей системы и их взаимодействия со специфичными системами и могут быть выполнены на всех уровнях тестирования. Проводится методом “черного ящика”.

Примеры:

* Позитивное тестирование;
* Негативное;
* Тестирование CRUD;
* Тестирование по сценариям использования.

**Нефункциональное** – тестирование, которое проводится для оценки характеристик систем и программ. Проверяется не корректность работы функций приложения, а сопутствующие характеристики (устойчивость к вредоносному проникновению, отказоустойчивость, выносливость при нагрузках, совместимость с другими браузерами и платформами (но не с другими системами)).

Примеры:

* Тестирование внешнего вида приложения (метод “ЧЯ”);
* Нагрузочное тестирование (метод “БЯ”);
* Тестирование безопасности (метод “ЧЯ”);
* Тестирование совместимости (метод “ЧЯ”).

**Структурное тестирование** – анализ и тестирование кода продукта, его архитектуры. Проводится методом “БЯ”.

Примеры:

* Unit-тесты;
* Интеграционные автоматизированные тесты;
* Тестирование веб-сервисов.

**Тестирование изменений** – это повторное тестирование уже протестированных программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в результате этих действий. Чаще проводится методом “ЧЯ”.

Примеры:

* Регрессионное тестирование;
* Тестирование, основанное на рисках

Тестирование, основанное на рисках, проводится путем повторного тестирования тех мест в приложении, в которые могут содержать баги. К таким местам могут относится места, где проводился рефакторинг, места, которые разрабатывались в условиях жестких сроков, места интеграции приложения с другими приложениями, места, где недавно исправлялись баги и любые другим места, критичные для бизнеса.

Виды функционального тестирования:

* Позитивное тестирование
* Негативное тестирование
* Исследовательское тестирование
* Интуитивное тестирование
* Тестирование по сценариям использования (ent-to-end testing)
* Тестирование, основанное на ролях (role-based testing)
* Инсталляционное тестирование
* CRUD тестирование

**Позитивное** – тестирование, при котором используются только валидные данные и выполняются только валидные действия.

**Негативное** – тестирование с использованием не валидных данные и действий, направленное на получение ошибок и предупреждений.

**Исследовательское** – подход к тестированию, который подразумевает под собой одновременно изучение приложения, проектирование тестовых сценариев и их немедленное выполнение.

**Интуитивное** – неподготовленное и недокументированное тестирование, “игра” с приложением по свободному сценарию без следования тестовой документации.

Исследовательское тестирование всегда имеет цель, интуитивное цели никогда не имеет.

**End-To-End тестирование** – подход, при котором проводится сквозное тестирование системы по сценариям использования системы реальными пользователями.

Сфокусировано на работоспособности приложения, переходы с одного в другой компонент.

**Role-Based тестирование** – тестирование, направленное на проверку ограничения прав доступа к конкретному функционалу на основе роли и привилегий пользователя.

Идет упор именно на использование ролей и прав доступа.

**Инсталляционное тестирование** – тестирование правильности инсталляции и настройки, а также обновления или удаления ПО.

Примеры проверок:

* Установка с параметрами по умолчанию
* Установка с измененными параметрами
* Установка на различные поддерживаемые платформы
* Установка в silent mode
* Установка обновлений на уже работающее ПО
* Удаление ПО

**CRUD - тестирование** корректности работы 4 функциональных возможностей каждой создаваемой записи в приложении.

* Создание
* Просмотр
* Редактирование
* Удаление

**Тестирование по приоритету** – виды тестирования, направленные на выявления качества функционала определенной важности

* Дымовое тестирование (smoke test);
* Тестирование критического пути (critical path test);
* Расширенное тестирование (extended test).

**Smoke test** – тестирование, направленное на определение на проверку самой главной, самой важной, самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования фичи и приложения.

**Critical path test** – тестирование, направленное на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности.

**Extended test** – тестирование, направленное на исследование всей заявленной в требованиях функциональности – даже той, которая имеет низкий приоритет.

**Лекция 5. Позитивное и негативное тестирование**

Модели поведения пользователя

**Пользователь-интуит.**

Пользователь не читал инструкций или не способен их прочитать. Находится несоответствие интерфейса программы существующим стереотипам.

**“Хороший” пользователь.**

Добросовестный пользователь действует в строгом соответствии с инструкциями. Поиск ошибок как в логике работы программы, так и в документации на программу.

**“Плохой” пользователь**

Недобросовестный пользователь стремится использовать программу непредусмотренным способом.

**Позитивное тестирование** – тестирование, при котором проверяются корректные варианты использования системы, а также реакция системы на валидные данные.

**Негативное тестирование** – тестирование, при котором проверяется варианты некорректного использования системы, а также реакция системы на невалидные данные.

Тестирование веб страницы или мобильного приложения идет слева направо сверху вниз. Если в задаче не сказано другого, то тестирование начинается с левого верхнего угла.

Для формы авторизации

**Позитивное тестирование**

* При открытии формы фокус должен быть в поле Username;
* Поле Password должно быть типа “password”
* Ввести существующий в системе Username, ввести правильный Password, нажать Log in
* Ввести существующий в системе Username, ввести правильный пароль нажать Enter
* Ввести Username, Password и нажать Cancel

**Негативное тестирование**

* Оставить поля пустыми и нажать “Enter” или Log in
* Ввести несуществующий в системе Username, ввести неправильный Password, нажать Log in

Для формы регистрации: Fullname

**Позитивное тестирование**

* Имя с пробелами
* Имя с дефисом и апострофом
* Имя в верхнем и нижнем регистре
* Имя в пределах минимума и максимума

**Негативное тестирование**

* Оставить поле пустым
* Имя за пределами минимума и максимума
* Спецсимволы вместо имени

Для формы регистрации: email

**Позитивное тестирование**

* E-mail в корректном формате
* Уникальный E-mail
* E-mail в пределах минимума и максимума

**Негативное тестирование**

* Оставить поле пустым
* Адрес, который не соответствует формату
* Спецсимволы вместо e-mail’a
* E-mail, который уже используется

Для формы регистрации: Username

**Позитивное тестирование**

* Username в корректном формате
* Уникальный username
* E-mail в пределах минимума и максимума

**Негативное тестирование**

* Оставить поле пустым;
* Username, который не соответствует формату;
* Username с пробелами;
* Username, который уже используется;
* Username, который за пределами минимума и максимума.

Для формы регистрации: Password

**Позитивное тестирование**

* Password отображается точками/звездочками
* Password в пределах минимума и максимума
* Password соответствует формату

**Негативное тестирование**

* Оставить поле пустым
* Password, который не соответствует формату
* Password, который за пределами минимума и максимума.

Для формы регистрации: кнопочки авторизации

**Позитивное тестирование**

* Нажать на кнопку соц сети, когда сессия в ней не открыта
* Нажать на кнопку соц сети, когда сессия в ней открыта
* Проверить возможность вернуться назад на форму регистрации

**Негативное тестирование**

* Попытаться зарегистрироваться одновременно, используя несколько соц сетей.

Для формы регистрации: календарь

**Позитивное тестирование**

* Выбор текущей даты
* Выбор даты в будущем
* Перелистывание месяцев вперед и назад
* Выбор корректной даты “Туда” и корректной даты “назад”

**Негативное тестирование**

* Оставить поле пустым
* Попытаться выбрать дату в прошлом
* Выбрать дату “Туда” в будущем и попытаться выбрать дату “Назад” раньше, чем “Туда”
* Выбрать корректно даты “Туда” и “Назад”, потом попытаться выбрать дату “Туда” позже, чем “Назад”
* Попытаться ввести дату с клавиатуры

Для формы регистрации день/месяц/год

**Позитивное тестирование**

* Выбор корректной даты в прошлом
* Ввод даты с клавиатуры
* Выбор текущей даты

**Негативное тестирование**

* Оставить поля пустыми
* Выбрать 31 число и месяц Апрель/Июнь/Сентября/Ноябрь
* Выбрать 29 февраля 1989 года
* Выбрать дату в будущем
* Набрать некорректную дату с клавиатуры

Для формы регистрации: телефон

**Позитивное тестирование**

* Ввод корректного номера телефона с пробелами
* Ввод корректного номера телефона с дефисами
* Ввод корректного номера телефона со скобками
* Ввод номера разных операторов
* Выбор другой страны (должен измениться код страны)
* Смена IP адреса (должна измениться страна по умолчанию)

**Негативное тестирование**

* Оставить поля пустыми
* Ввести спец символы или буквы
* Ввести телефон за пределами минимума и максимума
* Написать телефон, потом сменить страну
* Набрать номер, состоящий из нулей

Для формы регистрации: поле поиска

**Позитивное тестирование**

* Проверить, что placeholder исчезает, когда фокус находится в поле
* Поиск по нажатию на кнопку search
* Поиск по нажатию “Enter”
* Поиск существующих страниц
* Поиск с применением операторов
* Вставка текста поиска из буфера обмена
* Поиск по нескольким словам
* Autocomplete

**Негативное тестирование**

* Оставить поле пустым и нажать Search
* Поиск по спец символам
* Текст поиска за пределами минимума и максимума
* Поиск страниц, которой не существует

**Лекция 6. Исследовательское тестирование**

**Исследовательское тестирование** – это стиль в тестировании ПО, который предполагает сочетание личной свободы тестировщика и его обязанности постоянно оптимизировать качество своей работы путем восприятия изучения ПО, проектирования тестов и самого тестирования, как взаимодополняемых активностей, которые выполняются одновременно на протяжении всей разработки

Learning -> Test Design -> Test Execution -> Analysis -> Learning (Exploratory Testing).

Нет документации, практически нет планирования

Цели исследовательского тестирование:

* Получить понимание, как работает приложение и как оно выглядит
* Вынудить приложение показать все свои возможности
* Найти баги

Тестирование по сценариям:

“Фичи редко изолированы друг от друга. Чаще всего они используют одни и те же ресурсы, общие входные данные, функционирует на одних и тех же структурах данных. Поэтому тестирование каждой фичи по отдельности может на выявить багов, которые проявляются только, когда фичи взаимодействуют между собой. “James Whittaker,2010

Метафора исследовательского тестирования

* Бизнес район
* Исторический район
* Туристический район
* Развлекательный район
* Район отеля
* “Плохой” район

Бизнес район:

* The GuideBook Tour (Путешествие по путеводителю) предполагает что мы берем мануал и выполняем все действия, которые там описаны.
* The Blogger’s Tour (Путешествие блоггера) если человек пишет о этом, мы должны проверить работает либо оно так, как пишет человек.
* The Pundit’s Tour (Путешествие Недовольного Всезнайки) когда оставляют комментарий напр в play market мы проверим это действие
* The Money Tour ( Денежное путешествие) есть фича и именно оно будет проверяться(создание групповых чатов за деньги пр) Проверяем только те фичи, которые приносят деньги бизнесу
* The Skeptical Customer tour (Путешествие скептика) путешествие по денежным фичам, но с использованием альтернативных сценариев.
* The Landmark Tour (Путешествие по ориентирам) приложение определяются фичи и сценарии, которые должны хорошо работать и тестирование осуществляется только по ним.
* The Intellectual Tour (Путешествие интеллектуала) максимальное нагрузка на приложение
* The Arrogant American Tour (Путешествие заносчивого американца) делать максимально глупые и неоправданные действия
* The FedEx Tour (Путешествие почты FedEx) манипуляции с одним или несколькими элементами(сущностями) в системе
* The After-Hours Tour (Путешествие “после работы”) цель протестировать все задачи, которые выполняет приложение после своей основной работы (jobs, task)
* The Garbage Collector’s Tour (Путешествие мусорщика) тестируются фичи 2-го и даже 3-го приоритета

Исторический район:

* The Bad – Neighborhood Tour( Путешествие с “плохим” соседом) тестирование между старым и новым модулем системы
* The Museum Tour (Путешествие по музеям) тестирование старых давно используемых компонентов системы
* The Prior Version Tour (Путешествие по уже посещенным местам) мы проверяем как старые версии работают на новых платформах в новом окружении.

Туристический район:

* The Supporting Actor Tour (Путешествие суфлера) тестирование, используя альтернативные сценарии.
* The Back Alley Tour (Путешествие по переулкам) тестирование отдаленных, редко используемых мест в приложении.
* The All-Nighter Tour (Путешествие “по ночам”) нужно проверять длительное использование энергоемких операций.

Развлекательный район:

* The Collector’s Tour ( Путешествие коллекционера) максимальное создание различных сущностей
* The Lonely Businessman Tour (Путешествие одного бизнесмена) сделать действие максимально длинным действием.
* The Supermodel Tour (Путешествие супермодели) это проверка UI.
* The TOGOF Tour (Test One Get One Free) придумывая одну проверку, мы придумываем еще одну проверку, несколько действий на одну и ту же операцию.
* The Scottish Pub Tour (Путешествие по шотландским пабам) изучение всевозможных мест приложения

Район отеля:

* The rained-Out Tour (Испорченное путешествие) пр. при логине пароле нажимаете Cancel. Пройти приложение и проверить Cancel и Exit.
* The Couch Potato Tour (Путешествие ленивца) минимальное действие, чтобы совершить операцию.

“Плохой” район:

* The Saboteur (Путешествие саботажника) проводится подготовка, чтобы операция тестируемая не была выполнена.
* The Antisocial Tour (Антисоциальное путешествие) это негативное тестирование.
* The Obsessive-Compulsive Tour (Путешествие человека склонного к повторяющимся действиям) проверяется многократное нажатие на один элемент.

**Лекция 7. Тестирование требований**

**Требование** – это описание того, каике функции и с соблюдением каких условий должно выполнять приложение в процессе решения полезной для пользователя задачи.

**Формализация требований** – любая форма фиксации требований является валидной на проекте при условии, что все члены команды договорились о ее приемлемости.

Уровни и виды требований:

* Бизнес-требования
* Пользовательские требования
* Бизнес-правила
* Функциональные требования
* Нефункциональные требования
* Требования к интерфейсам
* Требования к данным

Бизнес-требования – выражают цели, ради которой разрабатывается продукт (зачем он нужен, какая от него ожидается польза). Результатом выявления требований на этом уровне является общее видение – документ, который как правильно представлен простым текстом и таблицами. Нет детализации поведения системы и иных технических характеристик, но могут быть определены приоритеты решаемых бизнес-задач, риски и т.п.

Бизнес требования:

* Нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.
* Необходимо в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену
* Нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров.

**Пользовательские требования –** описывает задачи, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на действия пользователя, сценарии работы пользователя) Т.к. появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объема работ, стоимости проекта, времени разработки и т.д.

* Варианты использования (user cases)
* Пользовательских историй (user stories)
* Пользовательских сценариев (user scenarios)

**Функциональные требования** – описывают поведение системы т.е. её действия (вычисления, преобразования, проверки, обработку и т.д.) В контексте проектирования функциональные требования в основном влияют на дизайн системы.

**Нефункциональные требования** – описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения. Здесь приводится более технические и детальное описание атрибутов качества. В контексте проектирования нефункциональные требования в основном влияют на архитектуру системы. (**ХАРАКТЕРИСТИКИ!**)

**Требования к интерфейсам** – описывает особенности взаимодействия разрабатываемой системы с другими системами и операционной средой.

**Требования к данным** – описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание азы данных и особенностей её использования.

Баги в требованиях:

* Противоречия
* Неоднозначность требования
* Неполное требование
* Отсутствие единого стиля в работе ПО
* Невозможность реализовать требования
* Отсутствие трассируемости

Неоднозначность требований – любые неоднозначность, которые находятся в описании требований

Отсутствие трассируемости – это когда в разных документах написаны разные требования к одному и тому же.

Способы уточнение требований:

* Интервью
* Анкетирование
* Наблюдение за фокус-группой
* Анализ похожих систем
* Мозговой штурм

Оформление вопросов:

* Вопросы пишите короткими и простыми
* Вопрос не должен содержать “или”
* Формулируйте вопрос так, чтобы ответ на него был максимально коротким
* Обдумывайте ответ, который можно получить
* Предлагая улучшения, подкрепляйте их фактами и весомыми доводами
* Просьбы подкрепляйте вежливыми оборотами

**Лекция 8. Нефункциональное тестирование**

Тестирование, которое проводится для оценки характеристик программного обеспечения. Проверяется не корректность работы функций приложения, а сопутствующие характеристики.

К характеристикам системы относится:

* Быстродействие
* Способность системы выдерживать большое количество пользователей
* Удобство использования
* Внешний вид приложения
* Эффективность приложения

Виды нефункционального тестирования:

* Тестирование производительности
* Тестирование безопасности
* Тестирование эргономичности (usability testing)
* Тестирование совместимости
* UI тестирование
* Тестирование локализации и интернационализации
* A/B тестирование
* Тестирование на отказ и восстановление
* Тестирование на соответствие стандартам
* Тестирование на прерывания (работы мобильного ПО)
* Тестирование соединения (работы мобильного ПО)

Т**естирование производительности** – комплекс тестов, определяющих поведение приложения при нормальной и предельной нагрузках.

Подвиды нагрузочного тестирования:

* Тестирование производительности
* Стресс-тестирование
* Нагрузочное тестирование
* Тестирование стабильности
* Тестирование на больших объемах данных

**Тестирование производительности** – тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определенной нагрузкой.

**Стресс-тестирование** – тестирования проводится для определения надёжности системы во время экстремальных или диспропорциональных нагрузок и отвечает на вопросы о достаточной производительности системы в случае, если текущая нагрузка сильно превысит ожидаемый максимум.

Меряет не только запредельную нагрузку, но и резкое одномоментное повышение количества пользователей.

**Нагрузочное тестирование** – тестирование, которое проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой.

Этой нагрузкой может быть, например, ожидаемое количество одновременно работающих пользователей приложения, совершающих количество одновременно работающих пользователей приложения, совершающих заданное число транзакций за интервал времени.

Такой тип тестирования обычно позволяет получить время отклика всех самых важных бизнес-транзакций. В случае наблюдения за БД, сервером приложения, сетью и т.д. этот тип тестирования может также идентифицировать некоторые узкие места приложения.

**Тестирование стабильности** – тестирование стабильности проводится с целью убедиться в том, что приложение выдерживает ожидаемую нагрузку в течение длительного времени.

При проведении этого вида тестирования осуществляется наблюдение за потреблением приложением памяти, чтобы выявить потенциальные утечки.

Кроме того, такое тестирование выявляет деградацию производительности, выражающуюся в снижении скорости обработки информации и/или увеличении времени ответа приложения после продолжительной работы по сравнению с началом теста.

**Тестирование на больших объемах** – тестирование производительности системы на больших объемах данных.

**Тестирование безопасности** – комплекс тестов, определяющих степень уязвимости программного обеспечения к различным атакам.

**Тестирование эргономичности** – исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (такой как веб-страница, UI или устройство) для его предполагаемого применения. Это медом оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

Тестирование usability проводится путем фокус-группы в виде записи их действий и последующего интервью.

**Тестирование совместимости** – тестирование, целью которого является проверка корректной работы приложения в определенном окружении.

* Аппаратная платформа
* Сетевые устройства
* Периферия (принтеры, CD/DVD приводы…)
* ОС (Windows, Unix, MACOs)
* БД (MSSQL, MySQL, Oracle)
* Системное программное обеспечение (веб-сервер, firewall, антивирус)
* Браузеры (IE, Chrome, Mozila)

**UI тестирование** – тестирование, проверяющее соответствие внешнего вида продукта заявленным дизайнам и требованиям.

**Тестирование глобализации**

**Локализация ПО** – процесс адаптации ПО к культуре какой-либо страны. Как частность – перевод пользовательского интерфейса, документации и сопутствующих файлов программного обеспечения с одного языка на другой

**Интернационализация ПО** – технологические приемы разработки, упрощающие адаптацию продукта к языковым и культурным особенностям региона (регионов), отличного от ого, в котором разрабатывался продукт.

**Отличие локализации от интернационализации:**

Интернационализация производится на начальных этапах разработки, в то время как локализация – для каждого целевого языка.

В тестировании локализации мы не проверяем корректность перевода.

**Тестирование глобализации:**

* Наличия перевода во всем приложении
* Отображения букв/символов языков
* Корректность сортировки и фильтрации в разных локализациях
* Графическое отображение текста
* Отображение аудио и видео материалов
* Форматы даты, времени, чисел, валюты, меры весов, расстояния и т.д.

**A/B тестирование** – метод маркетингового исследования, суть которого заключается в том, что контрольная группа элементов сравнивается с набором тестовых групп, в которых один или несколько показателей были изменены, для того, чтобы выяснить, какие из изменений улучшают целевой показатель.

**Тестирование на отказ и восстановление** – тестирование, которое проверяет продукт с точки зрения способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками программного обеспечения, отказами оборудования или проблемами связи. Целью данного вида тестирования является проверка систем восстановления, которые, в случае возникновения сбоев, обеспечат сохранность и целостность данных тестируемого продукта.

**Тестирование по стандартам** – процесс тестирования для определения соответствия компонента или системы стандартам, нормам и правилам.

* ISO/IEC 25010:2011
* Apple guidelines
* Google Play guidelines

**Тестирование на прерывания** – тестирование мобильного программного обеспечения на корректность обработки различных прерываний в работе приложения.

* Входящие звонки
* СМС сообщения
* Системные сообщения (напр. Низкий уровень заряда батареи)
* Screen-lock
* Уход в спящие режим

**Тестирование соединения** – тестирование мобильного программного обеспечения на корректность работы на различных типах соединений

* 2g
* 3g
* 4g
* wi-fi
* EDGE
* No connection

**Тестирование утечки памяти**

**Лекция 9. Автоматизация тестирования**

Часть процесса тестирования на этапе контроля качества в процессе разработки программного обеспечения. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогают сократить время тестирования и упростить его процесс. Основное назначение – сокращение времени.

**Уровни авто-тестирования:**

* Уровень модульного тестирования (unit tests)
* Уровень функционального тестирования (non-UI tests)
* Уровень тестирования через пользовательский интерфейс (UI tests)

**Модульное тестирование** – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

**Правила Unit-Тестов**

Тесты должны быть:

* Достоверными
* Не зависеть от окружения, на котором они выполняются
* Легко поддерживаться
* Легко читаться и быть простыми для понимания (даже новый разработчик должен понять, что именно тестируется)
* Соблюдать единую конвенцию именования
* Запускаться регулярно в автоматическом режиме.
* Один тест должен проверять только одну сущность
* Тесты должны храниться в системе контроля версий
* Названия методов должны быть “говорящими”

**Non-UI тестирование** – процесс в программировании, позволяющий проверить работоспособность приложения используя программный интерфейс приложения.

**UI-тестирование** – процесс в программировании, позволяющий проверить работоспособность и внешний вид приложения, используя графический интерфейс приложения.

Инструмент –> Браузер -> Сервер

**Подходы в UI автоматизации:**

* Data-driven подход
* Keyword-driven подход
* Behavior-driven подход

**Области для автоматизации:**

* Труднодоступные места в системе
* Часто используемая функциональность, риски от ошибок в которой достаточно высоки
* Рутинные операции
* Длинные end-to-end сценарии
* Проверка данных, требующих точных математических расчетов

**Процесс автоматизации тестирования:**

* Подготовка и планирование
* Дизайн и разработка
* Прогон тестов и сопровождение

**Подготовка и планирование:**

* Анализ требований клиента к автоматизации
* Анализ тесткейсов для ручного тестирования
* Оценка пригодности приложения к автоматизации (Приложение является не пригодным для тестирования, если в нем слишком много багов)
* Выбор инструмента автоматизации
* Создание пилотных скриптов
* Установка тестового окружения

К требованиям относятся: автоматизация существующих фич или новых фич, параллельно с разработкой, автоматизацией UI и Non-UI и какие есть требования средств по автоматизации.

Дизайн и разработка:

* Дизайн архитектуры (модули, фрэймворки)
* Подготовка тестовых данных (если будут использоваться)
* Определение повторно используемого кода и реализация многократного используемых функций, библиотек и фреймворков
* Разработка и отладка тестовых скриптов

Первый модуль, который есть это запуск приложения. Запуск приложения если мобильное, то скачать и установить билд, запустить приложение и либо создать пользователя этого приложения, либо с помощью sql(api) запросов.

Если веб -> старт браузера, очистка кэш и куки, создание пользователя путем api запроса или в самом веб приложении. Второй модуль - модуль с объектами страниц.

**Прогон тестов и сопровождение:**

* Запуск автоматических скриптов
* Анализ результатов запуска
* Создание отчетов о результатах запуска
* Поддержание тестов в актуальном состоянии

Скрипты могут выполняться либо после каждого комита.

**Лекция 10. Тестирование безопасности**

Тестирование безопасности – вид нефункционального тестирования, направленный на оценку уязвимости программного обеспечения к различным атакам.

Уязвимость – это недостаток или слабость в проектировании, реализации, эксплуатации или управлении системы, которые могут быть использованы для компрометации целей безопасности системы.

Угроза – это что угодно (вредоносный внешний злоумышленник, внутренний пользователь, нестабильность системы и т.д.), что может нанести ущерб частям приложения (ценным ресурсам, как бд фс).

**ТБ** – это ряд активностей, нацеленных на демонстрацию соответствия приложения требованиям к безопасности, предъявляемым владельцами бизнеса.

Эффективный процесс тестирования должен включать следующие компоненты:

* Люди – убедиться, что люди обучены и проинструктированы;
* Процессы – убедиться, что есть адекватные стандарты и стратегии обеспечения качества;
* Технологии – убедиться, что процесс показал себя, как эффективный в процессе его внедрения и выполнения.

**Техники тестирования безопасности:**

* Ручное инспектирование и ревью – анализ первоначальных требований к приложению
* Моделирование угроз
* Ревью кода
* Тестирование на проникновения – sql injection и др.

**Эффективный процесс тестирования**:

* Перед началом разработки
* Определение жизненного цикла разработки ПО
* Ревью стратегий и стандартов безопасности
* Определение параметров и метрик в обеспечении безопасности
* Во время создания требований/дизайна
* Ревью требований к безопасности
* Ревью архитектуры и дизайна
* Создание и ревью UML моделей
* Создание и ревью моделей угроз
* Во время разработки
* Поверхностный просмотр кода
* Код ревью
* Во время деплоймента
* Тестирование на проникновения
* Тестирование конфигурации приложения
* Поддержка приложения
* Периодическое ревью процесса
* Периодическое тестирование на проникновения

ТБ делится на 2 фазы:

* Пассивная фаза (сбор информации, изучение приложения)
* Активная фаза

**ТБ активная фаза:**

* Сбор информации
* Тестирование конфигурации
* Тестирование ролей и прав доступа
* Тестирование аутентификация
* Тестирование авторизации
* Тестирование состояния сессий
* Тестирование входных данных
* Обработка ошибок
* Криптография
* Тестирование бизнес логики
* Тестирование клиентской части ПО

**Сбор информации:**

**Поиск утечки информации с помощью поисковиков**

Цель: понять, какая конфиденциальная информация о дизайне и конфигурации приложения стала доступная непосредственно (на сайте организации) или косвенно (на стороннем веб-сайте)

**Определение веб-сервера:**

Цель: найти версию и тип работающего веб-сервера для определения известных уязвимостей и соответствующих экспойтов для использования во время тестирования.

**Ревью метафайлов веб-сервера:**

Цель: найти утечки информации о структуре приложения (файлов или папок), а также найти список файлов/папок, которые избегаются краулерами поисковых систем.

**Поиск приложений на веб-сервере:**

Цель: путем прослушивания портов, а также подстановкой названий приложений в URL, найти другие приложения, которые могут быть уязвимы.

**Ревью комментариев и мета-данных:**

Цель: найти утечки информации в комментариях кода и мета данных.

**Идентификации входных точек:**

Цель: понять, как формируются запросы и типичные ответы от приложения.

**Определение web-application Фреймворка:**

Цель: найти версию и тип работающего web-application фреймворка для определения известных уязвимостей и соответствующих эксплойтов для использования во время тестирования

Определение архитектуры приложения:

* Firewall
* Load balancers
* Etc

**Тестирование сети/инфраструктуры:**

* Тестирование известных багов веб-серверов и серверов приложения
* Тестирование административных инструментов (которые участвую в работе приложения)

**Тестирование платформы приложения**

Хранение сенситивных данных в системных файлах, логах и т.д.

**Тестирование содержания хранимых файлов:**

* Содержимое .asa I .inc файлов
* Содержимое txt файлов .bak .old и других файлов на сервере.

Тестирование HTTP методов:

* PUT: метод позволяет загрузить файлы (вредоносные) на сервер
* DELETE: метод может быть использован для удаления критически-важных для работы файлов
* CONNECT: этот метод мог позволить клиенту использовать веб-сервер как прокси
* TRACE: этот метод отражает клиенту то, что было отправлено на сервер. Может быть использовано для атак, известных как Cross Site Scripting.

**Тестирование ролей и прав доступа**

Цель: проверить доступность определенных компонентов и функциональностей для различных ролей в приложении.

**Тестирование процесса регистрации пользователя**

Цель: проверить процесс регистрации на соответствие требованиям к безопасности ПО.

**Тестирование аутентификации**

**Тестирование передачи логина/пароля**

* Отправка данных используя POST запрос через HTTP протокол
* Отправка данных используя POST запрос через HTTPS протокол
* Отправка данных используя POST запрос через HTTPS протокол, но используя HTTP протокол
* Отправка данных используя GET запрос через HTTPS протокол

**Лекция 11. Тестовая документация**

**Тестовая документация** – это документация, создаваемая тестировщиками, которая помогает в выполнении различного рода активностей в рамках тестирования программного обеспечения.

Помимо ТД, есть еще проектная документация. Ее могут создавать любые члены команды. К ней относится: план работы, график работы, написание функциональных и нефункциональных требований, coding guidelines.

**Планирование тестирования:**

* Тест план
* График тестирования
* Матрица устройств
* Матрица прослеживаемости
* Тестовый набор
* Тест сценарии
* Тест кейсы
* Чеклист

**Отчетность:**

* Отчеты об ошибках
* Отчеты о результатах тестирования

Ни один из этих видов не является обязательным, но чем проект и продукт больше, тем больше разнообразие тестовой документации.

**Тест план** – документ, описывающий весь объем работ по тестированию, начиная с описания тестируемых объектов, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.

**Цель написания тест плана:**

* Продумать стратегию тестирования программного обеспечения
* Описать процесс тестирования на проекте, и как он встраивается в процесс разработки
* Обеспечить информированность каждого члена команды об активностях QA команды, распределении обязанностей и зон ответственности
* Скорректировать ожидания заказчика от команды тестирования.

**График тестирования** – документ, описывающий последовательность выполнения активностей по тестированию членами QA команды, с указанием дат начала выполнения работ и их завершения.

Составляется, если команда от 2 и более человек.

Цель создания графика тестирования:

* Согласовать работу команды разработки и тестирования
* Обеспечить информированность каждого члена команды о последовательности задач, а также о сроках их выполнения
* Обеспечить прозрачность процесса тестирования для заказчика
* Обеспечить возможность отслеживания отставаний от плана и влияния добавления дополнительных задач команде.

**Матрица устройств** – документ, используемый на проектах, где разрабатываются мобильные приложения, который описывает конфигурацию устройств, где будет проводиться тестирование.

**Цель создания матрицы устройств:**

* Определить оптимальное количество устройств, необходимых для тестирования
* Убедится, что устройства с различными версиями необходимого оборудования участвуют в тестировании.
* Сосредоточить усилия тестировщика на устройства, оптимально полно покрывающих требования проекта
* Обеспечить прозрачность тестирования для заказчика.

**Матрица прослеживаемости** – документ, используемый для определения покрытия требований тестовой документацией. Это таблица, в которой ID требований соответствует id документации.

Цель создания матрицы прослеживаемости:

* Обеспечить должное покрытие всех функциональных и нефункциональных требований тестами.

**Тестовый набор** – документ, вмещающий в себя набор тестов/тестовых случаев/тестовых сценариев.

Тестовый набор формируется для определенных видов или уровней тестирования.

Цель создания тестового набора:

* Сгруппировать тестовые случаи по:
  + Видам тестирования
  + Уровням тестирования
  + Приоритету
* Облегчить распределение объема тестирования в команде
* Облегчить оценку трудозатрат тестировщика – это оценка, сколько времени понадобится на выполнение задачи.

**Тест сценарий** – документ, описывающий последовательность шагов, которые необходимо выполнить тестировщику, с целью выявления дефектов в приложении.

Виды тестирования, для которых пишут сценарии:

* Исследовательское тестирование;
* End-to-end тестирование;
* Role-based тестирование.

Тестовый сценарий может содержать в себе 30+ шагов.

**Тест кейс** – документ, описывающий последовательность шагов и ожидаемый результат. Документ направленный на проверку атомарных требований.

Виды тестирования, для которых пишут тест кейсы:

* Позитивное тестирование;
* Негативное тестирование;
* CRUD тестирование.

**Чеклист –** документ, перечисляющий идеи для проверки. Документ, который очень поверхностно указывает, что необходимо проверить в приложении, но не указывает, как это сделать.

Виды тестирования, для которых пишут чеклисты:

* Тестирование совместимости
* Тестирование инсталляции продукта
* Исследовательское тестирование
* CRUD тестирование
* Тестирование на прерывания и т.д.

**Отчет об ошибке** – документ, описывающий проблему, найденную в процессе тестирования в приложении.

**Отчет о результатах тестирования** – документ, описывающий результаты проведенного уровня или вида тестирования, который дает ясное понимание о количестве и серьезности найденных ошибок, количестве пройденных успешно и неуспешно тестов, а также о качестве самого приложения.

**Лекция 12. Техники тест дизайна**

**Тест дизайн (тд)** – этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест кейсы, чеклисты или тест сценарии), в соответствии с определенными ранее критериями качества и целями тестирования. Проверить на совместимость, проверить что функционал работает, проверить как приложение справляется с не валидными действиями.

На старте проекта команда и заказчик договариваются по каким критериям каждая из сторон могут понять, что приложение достаточно качественное и готов в релиз. (100% тест кейсов должно быть пройдено и 90% из них успешно (без багов)).

Цель техник ТД: определить тестовые условия, тестовые случаи (проверки) и тестовые данные.

Категории техник ТД

* Разработка тестов методом черного ящика
* Метод создания тестов на основе опыта
* Разработка тестов методом белого ящика

Методы на основе опыта.

**Метод черного ящика**

Техника базируется на анализе спецификаций или других проектных документов, описывающих требования к системе.

**Метод белого ящика**

Техника базируется на анализе внутренней структуры компонентов системы.

Характеристики техник ТД

**Разработка тестов методом черного ящика:**

* Для описания задач, которые должны быть решены, программных продуктов или их компонентов, используются модели – формальные или неформальные;
* Из этих моделей систематически выводятся тестовые сценарии

**Разработка тестов методом белого ящика:**

* Тестовые сценарии выводятся на основе информации о том, как спроектировано программное обеспечение (например, на основе программного кода и подробного описания проектного решения).
* Для программного обеспечения может быть измерена величина покрытия для имеющихся тестовых сценариев, и последующие тестовые сценарии могут разрабатываться для систематического увеличения покрытия.

**Метод создания тестов на основе опыта:**

* Для определения тестовых сценариев используются человеческие знания и опыт.
* Знания тестировщиков, разработчиков, пользователей и заинтересованных лиц о программном продукте, его использовании и окружении, являются одним источником информации.
* Знания о вероятных дефектах и их распределении являются другим источником информации.

**ТД методом черного ящика**

* Эквивалентное разбиение
* Анализ граничных значений
* Тестирование таблицы решений
* Тестирование таблицы переходов
* Тестирование по сценариям использования

**Эквивалентное разбиение** – входные данные для программного обеспечения или системы разбиваются на группы, от которых ожидается сходное поведение, то есть они должны обрабатываться аналогичным образом.

Данные группы называются **классами эквивалентности.** Можно проверять только если есть поля для ввода.

**Анализ граничных значений**

Поведение на границах эквивалентных областей имеет наибольшие шансы быть некорректным, таким образом границы являются потенциальным источником дефектов. Минимальные и максимальные значения сегмента являются граничными значениям.

**Min -1**

**Min**

**Average**

**Max**

**Max+1**

Min-1 Max+1 все должны давать сообщения о не валидных действиях. Система не должна давать пользователю это сделать.

Example

**Min -1 65 >3 кредита**

**Min 66 > потребительский кредит**

**Average 80 > потребительский кредит**

**Max 99 > потребительский кредит**

**Max+1 100 > система не даст ввести больше 2 символов**

**Тестирование таблицы решений:**

Таблица решений содержит триггерные условия, обычно комбинации значений “истина” и “ложь” для всех входных условий, и результирующие действия для каждой комбинации условий. Каждый столбец таблицы соотносится с бизнес-правилом, определяющим уникальную комбинацию условий и результат выполнения действий, связанных с этим правилом.

Стандартом покрытия для тестирования таблицы решений обычно является наличие хотя бы одного теста для каждой колонки, что обычно включает в себя покрытие всех комбинаций триггерных условий.

Тестирование таблиц решений возможно только на формах, где есть фильтрация.

**Тестирование таблицы переходов:**

Система может показывать различные отклики в зависимости от текущих условий или предшествовавшей истории состояний. Данный метод позволяет тестировщику рассматривать систему с точки зрения её состояний, переходов между состояниями, входов или событий, активизирующих изменения состояний (переходы) и действия, к которым приводят эти переходы. Состояния системы или тестируемого объекта разделяемы, определяемы и конечны. Таблица состояний демонстрирует связи между состояниями и входами и может подсказать возможные некорректные переходы.

**По сценариям использования:**

Сценарий использования (use case) описывает взаимодействия между участниками (включая пользователей и систему) приводящие к полезным результатам для заказчика или пользователя системы. Сценарии использования могут быть описаны на уровне абстракций (бизнес сценарий использования, уровень бизнес-процессов, не связанный с технологией) или на системном ровне (сценарий использования системы на уровне системного функционала)

**Лекция 13.** **Тестовый случай**

Тестовый случай - формально описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определённой ситуации, определённых выходных данных.

* Цель написания тестовых случаев:
* Детально описать шаги тестирования и ожидаемый результат;
* Задокументировать требования;
* Облегчить передачу знаний по проекту;
* Подготовиться к автоматизации тестирования;

Атрибуты тестового случая:

* Идентификатор (ID);
* Краткое название тестового случая (Summary, Title)
* Цель тестового случая (Goal, Aim, Description)
* Предусловия (Precondition)
* Шаги (Steps);
* Ожидаемый результат (Expected result);
* Постусловия (Postcondition);
* Статус (Status);
* Уровень (Level);
* Приоритет (Priority);
* Автор (Author)
* Комментарии (Notes, Comments)

Статусы тестовых случаев:

* Не пройден (Not run);
* Успешно пройден (Passed);
* Неудачно пройденный (Failed);
* Заблокирован (Blocked);

Правила написания тс:

* 1 требование = 1 тестовый случай;
* Количество шагов не больше 7;
* В Steps / Expected Results – только то, что относится к цели тест кейса;
* Шаги, которые не относятся к цели тест кейса – это либо Preconditions, либо Post-Conditions;
* Минимум кликов – максимум результата;
* Тест кейсы должны идти последовательно, составляя сценарий;
* Тест кейсы должны быть разделены на смысловые части;
* Как можно меньше ссылок на сторонние документы;
* Без картинок.

**Лекция 14.** **Тестовые сценарии**

Тестовый сценарий – формально описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определенной ситуации, определенных выходных данных.

Тестовый случай

1. Проверяет одно требование
2. Проверяет функциональность и внешний вид
3. Содержит не больше 5-7 шагов
4. Каждый шаг содержит ожидаемый результат

Тестовый сценарий

1. Проверяет много требований
2. Проверяет только функциональность
3. Чем больше шагов, тем лучше
4. Ожидаемый результат не прописывается

Цель написания тестовых сценариев:

Подготовиться к проведению следующих видов тестирования

* End-to-end
* Exploratory testing
* Role-based testing

В end-to-end фигурирует 1 тип пользователя. Задача тестировщика: затронуть максимальное количество компонентов. Действия пользователя должны имитировать реальные бизнес сценарии использования приложения.

В сценариях для role-based тестирования должны использоваться разные типы пользователей, которые работают с одной и той же сущностью приложения.

Атрибуты тестового сценария:

* Идентификатор(ID)
* Краткое название(Summary)
* Описание сценария(Description)
* Участники(Actors)
* Предусловия(Precondition)
* Основной сценарий (Basic Flow)
* Альтернативный сценарий (Alternative Flow)
* Исключения (Exception Flow)
* Статус(Status)
* Приоритет(Priority)

Приоритет используется, чтобы в условиях сжатых сроков протестировать.

Статусы тестовых сценариев:

* Не пройден (Not run)
* Успешно пройден (Passed)
* Неудачно пройденный (Failed)
* Заблокирован (Blocked)

Basic flow – последовательность шагов или действий, направленных для достижения цели. Это прямой путь, при котором используются значения по умолчанию и выбираются действия, для которых система непосредственно предназначена. Описывается нумерованным списком. Иногда чтобы с ориентировать тестировщика, где он должен находиться после выполнения шага, вставляют шаг, что система показывает пользователю.

Alternative flow – последовательность шагов или действий, ведущая к достижению цели пользователя, но которая подразумевает отклонение от направления по умолчанию. Это всегда только позитивное тестирование.

Exceptional flow – последовательность шагов или действий, которая препятствует достижению цели и которая подразумевает отклонения от правильной работы приложения. Задача шага - это вызвать реакцию системы на не валидные действия пользователя.

**Лекция 15. Отчет о результатах тестирование**

Официальный документ, описывающий результаты, полученные в ходе проведения определенных видов/уровней тестирования.

Цель написания отчетов:

* Предоставить статистику по количеству проверок и результаты их выполнения;
* Предоставить статистику по количеству и серьезности найденных дефектов в приложении
* Повысить осведомленность команды о качестве текущей версии приложения.
* Обратить внимание команды и владельцев бизнеса на области, которые нуждаются в исправлении дефектов.
* Дать рекомендации относительно выпуска или наоборот задержки выпуска версии в релиз.(самое важное)

Структура отчета:

1. Уровень/вид проведенного тестирования(н)ужно указать номер build’a(
2. Количество пройденных проверок(test cases/test scenarios/checkpoints)

* Общее количество
* Количество успешно пройденных(passed)
* Количество неудачных(failed)
* Количество заблокированных проверок(blocked)

1. Количество найденных багов

* Общее количество
* Количество oritical багов
* Количество major багов
* Количество minor багов
* Количество trivial багов

1. Области/ компоненты, которые наиболее повреждены багами (больше 75% проверок failed)
2. Проблемы, с которыми столкнулись в процессе выполнения задания

Если это тестирование на совместимость, то мы указываем оборудование, в котором проверялась совместимость, платформа, версия платформа, браузеры.

В 3 пункте. Помимо количества, указывается названия багов + id.

В 4 пункте. Не имеет значения, какие баги были найдены.

Примеры: отсутствие доступа к среде тестирования, задержки с поставками build’ов, мнимые или противоречивая информация между требованиями и информацией в приложении, возобновленные баги.

Тест план. Test strategy

Тест стратегия – официальный документ, описывающий методологию тестирования, принятую в компании.

Одна и та же организация может иметь разные стратегии для разных продуктов, разных циклов разработки, разных уровней риска.

Стратегия сама по себе может состоять из нескольких видов:

* аналитические стратегии (примеры: тестирование, основанное на рисках, тестирование, основанное на требованиях)
* стратегии, основанные на моделях (примеры: тестирование, основанное на моделях использования приложения)
* Методические стратегии (примеры: тестирование по общепринятым стандартам (ISO 25010:2011), тестирование по стандартам, принятым в компании;
* Стратегии, соответствующие процессуальным нормам (примеры: тестирование по стандартам HIPPA, GDPR и т.д.)
* Реактивные стратегии (примеры: исследовательское тестирование)
* Консультативные стратегии (примеры: тестирование, основано на сценариях и данных предоставленных заказчиком ПО)
* Стратегии, исключающие регрессионное (повторное) тестирование (примеры: широкое использование автоматизации для любых повторяющихся тестов)

Разные стратегии могут быть объединены в одну для того, чтобы обеспечить максимальное достижение целей в тестировании, принятых в компании.

Тест стратегия включает:

* Описание процесса интеграции тестирования в процесс разработки
* Техники тест дизайна
* Методы и виды тестирования
* Уровни тестирования
* Обязательные и необязательные стандарты, которым должно соответствовать ПО
* Критерии начала и окончания тестирования
* Метрики, собираемые в процессе тестирования.
* Инструменты, используемые в тестировании
* Окружение, где проходит тестирование
* Процесс контроля качества и его метрики
* Дефект менеджмент – описание жизненного цикла багов
* Роли и обязанности членов команды тестирования.

Проектный тест план (Master TP) – официальный документ, описывающий стратегию тестирования, принятую на конкретном проекте. Если разрабатывается несколько приложений, Мастер Тест План должен описывать пересечение стратегий тестирования всех приложений.

Проектный тест план:

* Описывает, что будет тестироваться, а что нет
* Качественные характеристики, которые будут оцениваться, и которые не будут оцениваться (скорость отклика, удобство использования)
* График тестирования и бюджет (оплата лицензионных систем)
* Циклы тестирования и их соотношение с релизным планом проекта
* Взаимоотношения между командой разработки и тестирования, результаты работы (deliverables)
* Критерии начала и окончания тестирования.
* Риски в тестировании
* Общее руководство процессом тестирования на проекте
* Обязанности тестировщиков в рамках тестирования каждого продукта
* Тестовые данные и выходные данные после тестов.

Список того, что будет тестироваться включает в себя: компонент самого приложения, название приложений или продуктов, с которыми система будет интегрироваться. Список того, что не входит в тестирование, включает в себя: версии платформ и браузеров, которые не поддерживаются, интеграция со сторонними системами, которые будут тестироваться на стороне заказчика.

Риски в тестировании – это вероятность возникновения неблагоприятных ситуаций в процессе тестирования.

Общее руководство процессом тестирования на проекте – самое событие и действие команды при этой ситуации.

Level тест план – официальный документ, описывающий стратегию тестирования, принятую для конкретного уровня тестирования или вида тестирования. Состоит из графика тестирования, задач, дат окончания работ, и других моментов, которые отличаются или не описаны в Мастер Test Плане.