# ЛК – 1. ТЕСТИРОВАНИЕ ПО

**Тестирование ПО –** это \* поиск ошибок

\* проверка соотв. ПО требованиям и здравому смыслу

\* оценка работоспособности

\* способ контролировать качество.

Причины **ошибок** в ПО:

\* человеческий фактор

\* проблемы в описании требований к ПО

\* недостаток времени

\* недостаточно продуманная архитектура приложения

\* недостаточное знание бизнеса

\* нехватка навыков и опыта

\* изменения «в последнюю минуту»

## История тестирования ПО

**9 сентября 1945** – оформлен первый баг.

Первые программные системы разрабатывались в марках программ научных исследований или программ для нужд министерства обороны.

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

Тестирование выделялось в отдельный процесс, который начинается после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом. Чем выше риски, тем больше должно быть проверок.

**В 1960-х** много внимания уделялось «исчерпывающему» тестированию, которое должно проводится с использованием всех путей в коде или всех возможных входных данных.

Однако, это невозможно:

1. Количество возможных входных данных 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-х** годов тестирование ПО обозначалось как «процесс, направленный на демонстрацию корректности продукта» или как «деятельность по подтверждению правильности работы ПО».

Впоследствии этот метод тестирования был признан неэффективным.

ВО **второй половине 70-х** тестирование поменяло свое направление и представлялось как выполнение программы с намерением найти ошибки, а не доказать что она работает.

Успешный тест – это тест, который обнаруживает ранее неизвестные ошибки

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

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

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

В **начале 1990-х 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 текстами, ПО для проведения нагрузочного тестирования.

## Отличие тестирования от обеспечения качества

Когда мы тестируем, мы тестируем готовый продукт, а значит баги там уже есть. Все, что мы можем сделать – это контролировать, т.е.дать понять всем, сколько багов мы нашли.

Обеспечение качества заключается в том, что мы предпринимаем какие-то действия для того, чтобы багов не было.

Активности, которые помогают обеспечить качество:

\*тестирование требований

\*unit-тесты

\*code review

\*ранняя интеграция

И только первая активность делается тестировщиками.

ISO – International Organization of 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 заявления 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 подхарактеристики.

## Модель качества

1. **Модель качества при использовании**

* **Эффективность (результативность)** – точность и полнота, с которой пользователи достигают поставленных целей
* **Производительность** – связь точности и полноты достижения пользователями цели с израсходованными ресурсами
* **Удовлетворенность:**
* **Полноценность** – пригодность к полезности
* **Доверие** – степень, с которой пользователь доверяет приложению
* **Удовольствие**
* **Комфорт** (физический)
* **Свобода от риска** (смягчение отрицательных последствий)
* **Экономического риска**
* **Риска для здоровья и безопасности**
* **Экологического риска**
* **Покрытие контекста**
* **Полнота контекста –** степень, в которой ПО применимо при использовании на маленьком экране с низкой сетевой пропускной способностью неквалифицированными пользователями или в отказоустойчивом режиме
* **Гибкость** – степень, в которой продукт может быть использован с эффективностью, результативностью и свободой от риска, в соответствии с требованиями в условиях, выходящими за рамки первоначально определенных.

1. **Модель качества продукта**

* **Функциональная пригодность** – степень, в которой продукт обеспечивает выполнение функций в соответствии с заявленными требованиями
* **Функциональная полнота** – степень покрытия функций всех определенных задач и целей пользователя.
* **Функциональная корректность** – когда приложение делает то, что оно должно делать.
* **Функциональная целесообразность** – степень функционального упрощения выполнения задач пользователя.
* **Уровень производительности –** степень производительности ПО относительно суммы использования ресурсов эффективности.
* **Временные характеристики**
* **Использование ресурсов**
* **Потенциальные возможности** – степень соответствия требованиям предельных значений параметров продукта
* **Совместимость**
* **Сосуществование** – степень, в которой ПО может сосуществовать с другим ПО на одном девайсе, сервере или в одной системе
* **Интероперабельность** – степень, в которой две и более системы способны обмениваться информацией, использовать информацию друг друга
* **Удобство использования**
* **Определимость пригодности** – то, насколько быстро пользователь понимает, что приложение поможет ему достигнуть цели
* **Изучаемость** – насколько доступно и понятно изложен материал в обучающих приложениях
* **Управляемость –** можно управлять своими данными
* Защищенность от ошибки пользователя
* Эстетика пользовательского интерфейса
* **Доступность** – возможность использования системы для достижения определенной цели в указанном контексте использования широким кругом людей с самыми разными возможностями.
* **Надежность**
* **Завершенность** (зрелость) – приложение отрабатывает все возможные варианты, связанные с надежностью системы.
* **Готовность** (доступность) – можно оценить как долю общего времени в течении которого приложение находится в работоспособном состоянии.
* **Отказоустойчивость –** система должна сама себя восстановить в случае сбоя.
* **Восстанавливаемость** – способность продукта восстановить данные или самого себя.
* **Защищенность**
* **Конфиденциальность** – это обеспечение доступа и ограничение доступа к определенным ресурсам приложения в соответствии с ролью пользователя.
* **Целостность** – это степень предотвращения системой несанкционированного доступа или модификация данных.
* **Неподдельность** – степень, в которой может быть доказан факт события или действия таким образом, что этот факт не может быть опровергнут.
* **Отслеживаемость** – степень, до которой действия объекта могут быть прослежены однозначно.
* **Подлинность** – степень достоверности того, что объект или ресурс равны требуемому объекту или ресурсу.
* **Сопровождаемость**
* **Модульность**
* **Возможность многократного использования**
* **Анализируемость** – насколько быстро можно проанализировать и понять где ошибка.
* **Модифицируемость** – это непосредственно связано с исправлением ошибок.
* **Тестируемость** – можно протестировать
* **Переносимость**
* **Адаптируемость**
* **Устанавливаемость** – система должна устанавливаться на предусмотренные окружения, нормально обновляться и удаляться, возможность изменить папку установки.
* **Взаимозаменяемость** – способность продукта заманить другой конкретный программный продукт для достижения тех же целей в тех же условиях.

1. **Модель качества данных ISO/IEC 25012:2008**

## Терминология

**Dev Environment** – среда, где разработчики отслеж, проверяют свои изменения.

**QA Environmen**t – среда, где тестировщики проверяют качество приложения.

Отличие в том, что на Dev environment происходят частые изменения, а на QA 1 раз в день.

**Staging Environment** – среда, где имитируется работа приложения, как на Production Environment.

**Production Environment** – среда, где работают конечные пользователи приложения.

**Verification** – процесс тестирования приложения на соответствие предъявленным требованиям, которое осуществляется на протяжении всей разработки приложения.

**Validation** – проверка приложения на соответствие бизнес-цели. Проводится, как правило, перед релизом.

# ЛК – 2. ПРОЦЕСС РАЗРАБОТКИ ПО

**Процесс разработки ПО** – это структура, согласно которой построено создание нового ПО.

**Жизненный цикл ПО** – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Составляющие процесса:

* Требования
* Архитектура
* Дизайн
* Имплементация
* Тестирование
* Релиз

Модели разработки ПО – структура, систематизирующая различные виды проектной деятельности, их взаимодействие и последовательность в процессе разработки ПО.

* Каскадная модель
* V-модель
* Итеративная модель
* Инкрементальная модель
* Спиральная модель

### Каскадная модель

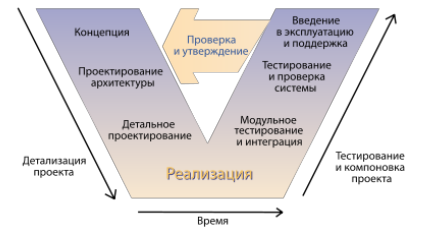
В современной модели на каждом этапе + тестирование.

Подходит: медицинское, военное ПО; крупные проекты; распределенная команда.

Не подходит: часто меняются требования, небольшие проекты, небольшая команда.



### V-модель



### Итеративная модель

Подходит: небольшие проекты, небольшие команды, команда в одной локации.

Не подходит: распределенная команда, медицинское, военное ПО.

Всегда есть возможность в последнюю минуту сделать изменения!

требования

релиз дизайн

тестирование имплементация

**Agile –** философия разработки ПО.

**Agile-манифест:**

* Люди и взаимодействие важнее процессов и инструментов
* Работающий продукт важнее исчерпывающей документации
* Сотрудничество с заказчиком важнее согласования условий контрактов
* Готовность к изменениям важнее следования первоначальному плану.

# ЛК – 3. ПРОЦЕСС ТЕСТИРОВАНИЯ ПО

Процесс тестирования ПО состоит из **5 последовательных** стадий:

\* планирование и управление тестированием

\* анализ и проектирование

\* внедрение и реализация

\* оценка критериев выхода и создание отчетов

\* действия по завершению тестов

## 1. Планирование и управление тестированием

**Планирование тестированием** – действия, которые направленны на определение целей тестирования и описание задач тестирования для достижения .этих целей.

**Управление тестированием** – постоянное сопоставление текущего положения дел с планом и отчетность о состоянии дел, включая отклонения от плана (двигаемся ли мы к пост.цели, есть ли отклонения от плана, успеваем нет)

**Планирование тестирования ПО:**

1. анализ требований

(что требуют от нас и приложения)

1. определение целей тестирования

(найти баги, шаги воспроизведения и др)

1. определение общего подхода к тестированию

(уровни и виды тестирования, критерии входа)

1. интегрирование с разработкой ПО

(требования, архитектура, дизайн, разработка, тестирование, релиз)

1. решение, какие роли нужны для выполнения тестирования. Когда и как производить тестирование и как оценивать результаты.
2. составление графика тестирования
3. определение шаблонов для тестовой документации
4. выбор *метрик* для мониторинга и контроля подготовки и проведения тестирования, исправления дефектов, проблем и писков.

**Критерии входа** – определенные критерии, по которым мы понимаем когда нужно начать тестирование.

* + готовность и доступность тестового окружения (квей енвиромень)
  + готовность средства тестирования в окружении (tools, soft)
  + доступность тестируемого кода
  + доступность тестовых данных

## 2. Анализ и проектирование

это деятельность, во время которой общие цели тестирования материализуются в тестовые условия и тестовые сценарии.

В это время мы можем выполнить следующие действия :

* рецензирование базиса тестирования (знакомимся с функц/нефункц.треб., архитектура, дизайн, технические требования к интерфейсу)
* оценка тестируемости базиса тестирования и объектов тестирования
* идентификация и расстановка приоритетов тестирования.
* выявление необх.д-х для поддержки тестовых условий и тестовых сценариев
* проектирование и установка тестового окружения и выявление необходимой инфраструктуры и инструментов
* создание 2-направленной трассируемости между тестовым базисом и тестовым сценарием (каждое требование протестировано)

## 3. Внедрение и реализация тестов

Это деятельность, где процедуры тестирования или автоматизированные сценарии задаются последовательностью тестовых сценариев, а также собирается любая информация, необходимая для выполнения тестов, разворачивается окружающая среда и запускаются тесты.

**Что еще можем делать?**

* завершение, реализация и расстановка приоритетов тестовых сценариев

(включая проектирование тестовых данных)

* написание автоматизированных сценарием тестирования
* проверка правильности настройки тестового окружения
* проверка и обновление 2-направленной трассируемости между тестов 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 в баг-трекинговую систему
* повторное тестирование областей, где были исправлены ошибки и областей, где могут появится новые ошибки после исправления уже известных ошибок (регрессионное тестирование)

## 4. Оценка критериев выхода и создание отчетов

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, определенными в плане тестирования;
* анализ необходимости использования дополнительных тестов или изменения критериев выхода;
* написание итогового отчета о тестировании для заинтересованных лиц.

**Критерий выхода** определяет когда нужно прекращать тестирование.

\* степень покрытие кода, функциональности или рисков тестами;

\* оценку плотности дефектов или измерение надежности;

\* стоимость;

\* остаточные риски (направленные дефекты или недостаток тестового покрытия какой-либо области);

\* план, основанный на времени выхода ПО на рынок.

## 5. 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 разработки** |  **процесс тестирования**

анализ требований | анализ требований

разработка архитектура | планирование тестирования

проектирование дизайна | анализ и проектирование тестов

имплементация | реализация и выполнение тестов

отладка | реализация и выполнение тестов

| оценка критериев выхода и отчетность

инсталляция прилож. на production | действия по завершению тестирования

**процесс тестирования | процесс обеспеч.качества**

планирование и управление | тестирование требований ()

анализ и проектирование | реализация unit-тестов (разраб)

внедрение и реализация | реализ.интеграц.тестов (разр/автоматиз)

оценка критериев выхода и | code review (разраб)

создание отчетов | интеграция фичи в систему (разраб)

действия по завершению тестов | Alpha тестирование (команда тестир)

| отладка

| Beta тестирование (конечн.юзеры)

## Уровни тестирования ПО:

\* компонентное тестирование

(не юнит-тесты) (форма поиска, логин форма)

\* интеграционное тестирование

как фича интегрир.с другими фичами

\* системное тестирование

включ.все тесты (тестирование всей системы целиком)

\* приемочное тестирование

в процессе приемочного тестирования мы выполняем валидацию и проверяем на сколько приложение соотв.бизнес-целям.

верификация (компонент+интегр+системное) – провер единичн.требования

валидация (приемочное)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Каталог товаров | Поиск товаров | оплата | Профайл покупат | Компонентное |
| Каталог товаров + поиск товаров | | Оплата + профайл покупат. | | Интеграционное |
| Каталог товаров + поиск товаров + оплата + профайл покупателя | | | | Системное |

# ЛК - 4. ТИПОЛОГИЯ ТЕСТИРОВАНИЯ

## 2 метода тестирования существует:

* метод белого ящика – тестируется только код (бэкэнд приложения)
* метод черного ящика – тестирование через интерфейс программ (нет исходного кода)

**Типы тестирования** – группы активности тестирования, которые направлены на проверку работоспособности системы, где за основу принимаются различные цели и причины тестирования.

## Типы тестирования

### 1) функциональное тестирование

такое тестирование, которое разрабатывается на основе функций и возможностей системы, а также их взаимодействие с другими системами.

Проводится на разных уровнях;

Проводится методом черного ящика;

Примеры:

- позитивное тестирование

- негативное тестирование

- тестирование CRUD (create, read, update, delete)

### **2) нефункциональное тестирование**

проводится для оценки характеристик систем и программ

Проверяется не корректность работы функций приложения, а соответствующие характеристики.

Примеры:

- тестирование внешнего вида приложения (методом черного ящика)

- нагрузочное тестирование (методом черного и белого ящика)

- тестирование безопасности (методом черного ящика)

- тестирование совместимости (методом черного ящика)

### 3) структурное тестирование

анализ и тестирование кода продукта, его архитектуры.

Проводится методом черного ящика.

Примеры:

- unit-тесты

- интеграционные автоматизированные тесты

- тестирование веб-сервисов

### 4) тестирование изменений

любое повторное тестирование уже протестированных программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в результате этих действий.

Чаще проводится методом черного ящика.

Примеры:

- регрессионное тестирование

- тестирование, основанное на рисках

## Виды функционального тестирования

* позитивное тестирование
* негативное тестирование
* исследовательское тестирование
* интуитивное тестирование
* тестирование по сценариям пользователя (end-to-end testing)
* тестирование, основанное на ролях (role-based testing)
* инсталляционное тестирование
* CRUD тестирование

|  |  |  |
| --- | --- | --- |
| **Позитивное тестирование** | **Негативное тестирование** | |
| тестирование, при котором используются только валидные данные и выполняются только валидные действия | тестирование с использованием невалидных данных и действий, направленное на получение ошибок и предупреждений. | |
| **Исследовательское тестирование** | **Интуитивное тестирование** | |
| подход к тестированию, который подразумевает под собой одновременное изучение приложения, проектирование тестовых сценариев и их немедленное выполнение. | неподготовленное и недокументированное тестирование, «игра» с приложением по свободному сценарию без исследования тестовой документации. | |
| **Отличие** исследовательского тестирования от интуитивного:  \* у исследовательского тестирования всегда есть цель. Например, проверить, что новые компоненты или фича работает корректно; приложение, оставленное на ночь в рабочем состоянии, продолжает работать корректно и т.д.  \* у интуитивного тестирования никогда нет цели. | | |
| **End-to-end тестирование** | **Role-based тестирование** | |
| подход, при котором проводится сквозное тестирование системы по сценариям использования системы реальными пользователями | тестирование, направленное на проверку ограничения прав доступа к конкретному функционалу на основе роди и привилегий пользователя | |
| Основная **задача тестировщика** – продумать как можно больше разнообразных сценариев использования приложений.  Существенная **разница** между этими 2 видами тестирования: первое имеет одного пользователя и затрагивает как можно больше компонентов приложения, в во втором – как можно больше ролей. | | |
| **Инсталляционное тестирование** | **CRUD тестирование** | |
| тестирование, при котором проверяется корректность инсталляции и настройки приложения, а также обновления и удаления ПО. | Тестирование корректности работы 4 функциональных возможностей каждой создаваемой записи в приложении:  - создание  - просмотр  - редактирование  - удаления | |
| Примеры инсталляционного тестирования:  \*установка с параметрами по умолчанию  \*установка с измененными параметрами  \*установка в silent mode  \*установка на разл.поддерж-мые платформы  \*установка обновлений на работающее ПО  \*удаление ПО | | Создание и редактирование всегда должно происходить по тем же правилам.  **2 вида удаления**:  \*soft – скрывается, но ост.в БД  \*hard – запись полностью удал из БД |
| **Тестирование по приоритету** | **Smoke test** | |
| виды тестирования, направленные на выявление качества функционала определенной важности:  \*дымовое тестирование (самый важный)  \*тестирование критического пути(внизу  \*расширенное тестирование (extended) | тестирование, направленное на определение и проверку самой важной функциональности. Неработоспособность делает бессмысленной идею использования фичи или приложения. Баги, найденные в smoke test – **блокирующие.** | |
| **Critical path test** | **Extended test** | |
| тестирование, направленное на исследование функциональности, используемой типичными пользователями в типичной повседневной деятельности | тестирование, направленное на исследование всей заявленной в требованиях функциональности – даже той, которая имеет низкий приоритет. | |

# ЛК – 5. ПОЗИТИВНОЕ И НЕГАТИВНОЕ ТЕСТИРОВАНИЕ

**Метод** тестирования отвечает на вопрос «каким способом мы проводим тестирование»**.**

\*способ *белого ящика* – инспектируем код

\*способ *черного ящика* – взаимодействуем с системой через интерфейс

**Уровень** отвечает на вопрос «как глубоко мы тестируем систему».

\*на уровне компонента

\*на уровни интеграции

**Виды тестирования** отвечаю на вопрос «с какой целью мы тестируем, что мы хотим найти».

## Модели поведения пользователя

1. **пользователь-интуит**

пользователь не читал инструкций или не способен их прочитать. Находится несоответствие интерфейса/поведения программы существующим стереотипам

* проводим и позитивное, и негативное тестирование

1. **«хороший пользователь»**

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

* больший приоритет отдается позитивному тестированию

1. **«плохой» пользователь**

недобросовестный пользователь стремится использовать программу непредусмотренным способом

* негативное тестирование

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

Основной **целью** является проверка того, что при помощи системы можно делать то, для чего она создавалась.

## Примеры позитивного тестирования

В числовых полях есть разделители – мы можем отделять десятичные числа точкой или запятой.

* протестировать, что выбор високосного года проверяется корректно и не приводит к ошибкам и неисправным расчетам
* протестировать отрицательные числа в числовом поле (если разрешены)
* протестировать дробные числа в числовом порядке (если разрешены)
* протестировать, что система правильно обрабатывает деление на 0
* протестировать максимальную длину поля и что данные не обрезаются
* протестировать функциональность Timeout (web-сессия)
* протестировать сортировку (в таблице)

\* спецсимволы вверху (отсор.по 2-му символу) -> по алфавиту

\* сортировка по дате (проверить сортировку по дням, месяцам, годам).

* протестировать функциональность кнопок

\* кнопки должны делать то, для чего они предназначены

\* могут быть разные состояния кнопок

* проверить, что Privacy Policy & FAQ доступны для юзеров

\* Privacy Policy & FAQ находятся внизу и дб протестированы (ссылки)

* протестировать, что загруженные файлы можно открыть
* протестировать, что загруженные файлы можно скачать
* проверить все имейлы, которые генерирует система

\* проверить информацию внутри (что сообщение не пустое)

* проверить, что JS работает правильно во всех поддерживаемых браузерах (IE, Firefox, Chrome, Safari)
* подтверждающие сообщения (confirmation messages) должны использовать один и тот же СSS стиль и должен отличаться от стиля сообщений об ошибке (например, зеленый цвет)

\* когда красный – инстинктивно ошибка, зеленый – все хорошо

* проверяем тултипы (их значение и текст)

**\* тултип** – всплывающая подсказка

* проверить push-нотификации, текст в них и корректность работы
* первое значение в dropdown списке должны быть пустыми или ‘select’ (или другое соответствующее контексту)
* должны присутствовать опции select/deselect all для чекбоксов (при наличии более 10 чекбоксов)
* проверить надписи над полями
* проверить сортировку после добавления, апдейта или удалений какой-либо записи

## Негативное тестирование - противопоставление позитивному.

(неразрешенные данные и неразрешенные действия @Скиндер)

Основная **цель** – проверить, что система корректно обрабатывает невалидные данные/действия пользователя и в понятной пользователю форме дает обратную связь

## Примеры негативного тестирования

* протестировать, что все обязательные поля проверяются системой
* протестировать, что система не выдает валидационную ошибку на необязательные поля
* протестировать, что пользователь не может ввести больше/меньше символов, чем допустимо
* проверить, что если проходит какая-либо ошибка (404 или 500, или другая), то система перенаправляет пользователя на специальную страницу приложения, где находится предупреждающее сообщение (application crash or unavailable pages)
* протестировать, что случается, если пользователь удаляет куки после посещения сайта
* проверить расположение валидационного сообщения
* все сообщения об ошибках должны использовать один и тот же CSS стиль (красный цвет)
* ввод данных с пробелами в начале, середине и в конце

|  |  |  |  |
| --- | --- | --- | --- |
| **Позитивное тестирование** | | **Негативное тестирование** | |
| Тестирование страницы авторизации | | | |
| \* При открытии формы фокус дб в поле Username  \* Ввести существующий в системе Username, ввести правильный Password и нажать Log in  \* Поле Password должно быть типа «Password»  \* Ввести существующий в системе username и password, нажать Enter  \* Ввести username и password и нажать cancel | | \* Оставить поля пустыми, нажать Log in  \* Ввести несуществующий в системе Username, ввести неправильный Password, нажать Log In | |
| Тестирование полного имени | |
| \*имя с пробелами  \*имя с дефисом и апострофом  \*имя в верхнем и нижнем регистре  \*имя в пределах минимума и максимум | \*оставить поле пустыми  \*имя за пределами минимума и максимума  \*спецсимволы вместо имени |
| Тестирование поля e-mail | |
| \*e-mail в корректном формате  \*уникальный e-mail  \*e-mail в пределах минимума и максимума | \*оставить поле пустым  \*адрес, которые не соответствует формату  \*спецсимволы вместо e-mail  \*e-mail, который уже используется |
| Тестирование имени пользователя | |
| \*username в корректном формате  \*уникальный username  \*e-mail в пределах минимума и максимума | \*оставить поле пустым  \*username, который не соответствует формату  \*username с пробелами  \*username, который уже используется  \*username, который за пределами минимума и максимума |
| Тестирование пароля | |
| \*password отображается точками / звездочками  \*password в пределах минимума / максимума  \*password соответствует формату | \*оставить поле пустым  \*password, который не соответствует формату  \*password, который за пределами минимума и максимума |
| Тестирование footer’а (войти через соц.сети) | |
| \*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 | \*попытаться зарегистрироваться одновременно используя несколько социальных сетей |
| Тестирование даты (календаря) | |
| \*выбор текущей даты  \*выбор даты в будущем  \*перелистывание месяцев вперед и назад  \*выбор корректной даты «туда» и корректной даты «назад» | \*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 |
| Тестирование даты рождения | |
| \*выбор корректной даты в прошлом  \*ввод даты с клавиатуры  \*выбор текущей даты | \*оставить поля пустыми  \*выбрать 31 число и месяц апрель/июнь/ сентябрь/октябрь  \*выбрать дату 29 февраля 1898 года  \*выбрать дату в будущем  \*набрать некорректную дату с клавиатуры |
| Тестирование номера телефона | |
| \*ввод корректного номера телефона с пробелами  \*ввод корректного номера телефона с дефисами  \*ввод корректного номера телефона со скобками  \*ввод номера телефона разных операторов  \*выбор другой страны (должен изменится код страны)  \*смена IP адреса (должна изменятся страна по умолчанию) | \*оставить поля пустыми  \*ввести спецсимволы или буквы  \*ввести телефон за пределами минимума и максимума  \*написать телефон, потом сменить страну  \*набрать номер состоящий из нулей |
| Тестирование поля поиска | |
| \*проверить, что placeholder исчезает, когда фокус находится в поле  \*поиск по нажатию на search  \*поиск по нажатию на enter  \*поиск существующих страниц  \*поиск с применением операторов  \*вставка текста поиска из буфера обмена  \*поиск по нескольким словам  \*autocomplete | \*оставить поле пустым и нажать search  \*поиск по спецсимволам  \*текст поиска за пределами минимума и максимума  \*поиск страниц, которых не существует |

# ЛК – 6. ИССЛЕДОВАТЕЛЬСКОЕ ТЕСТИРОВАНИЕ

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

## Цели исследовательского тестирования:

* проверить систему с т.зр. конечного пользователя;
* найти баги, которые не были найдены путем формального процесса тестирования;
* изучить работу приложения, опираясь не только на документацию, но и на поведение системы после манипуляций пользователя.

## Процесс исследовательского тестирования:

* 1. Подготовка
     1. уточнение feature under test
     2. уточнение цели тестирования
     3. уточнение окружения (платформы/браузера)
     4. уточнение продолжительности сессии
     5. уточнение, какая документация необходима до/после тестирования
  2. Тестирование/оформление документации
  3. **Дебрифинг** (de-briefing) – краткий рассказ заинтересованным лицам о том, как проходило тестирование, какие проверки выполнялись, с какими проблемами столкнулись, на какие области в будущем следует обратить внимание.

**Mission** в исследовательском тестирования – это описание цели сессии исследовательского тестирования: на что тестировщик планирует обращать внимание, какой аспект фичи/приложения планирует проверять.

Продолжительность сессий:

* Short (60 min)
* Normal (90 min)
* Long (120 min)

## Документация в исследовательском тестировании:

* checklist (общий, краткий список с проверками)
* test charter (подобный документ с деталями тест-сессии)
* записи шагов тестировщика в приложении (с помощью ПО (Jira Capture, Session teste, qTrace))
* bug reports (отчеты об ошибках)
* логи приложения (если уместно)

## TEST CHARTER : структура документа

* test charter **id** – уникальный идентификатор, которые не используется нигде
* **charter name** – краткое название
* **mission** – цель сессии (3-5 предложений)
* **areas** – список компонентов/модулей, которые будут затронуты тестированием
* **test environment & URL** – ссылка на application under test, ОС/брайзеры/устройства, где будет проходить тестирование
* **start date & time**
* **tester name** – свое имя и фамилия
* **duration** (short, normal, long и количество минут)
* **testing notes** – список проверок, что планируется проверять
* data files–если требуется
* **bugs** – названия багов и их ID
* issues – описание трудностей при тестировании
* session setup time – время, потраченное на подготовку (установку нового билда на мобильное устройство)
* test design & rest execution – % потраченный на продумывание проверок и на само тестирование

то, что жирным – обязательно!

Исследовательское тестирование ПО – Джеймс Уиттакер, 2010г – полезные советы, техники для тест дизайна

**Тур** – сессия исследовательского тестирования, которая имеет цель.

**Район** – группы туров, разделенные по определенной тематике.

## Метафора исследовательского тестирования:

* бизнес район – все туры, которые связаны с бизнесом приложения
* исторический район – связан с прошлыми версиями или областями приложения, которые давно не тестировались
* туристический район – туры, не связанные с основными целями в приложении
* развлекательный район – фичи, которые носят второстепенный характер и имеют более развлекательную цель (отзывы на товары, комменты, лайки)
* район отеля – тур, который связан с минимальным количеством действий пользователя (пользователь не хочет напрягаться)
* «плохой» район – любые вредоносные действия пользователя

## ТУРЫ РАЙОНОВ

|  |  |
| --- | --- |
| **Район** | **Описание тура** |
| БИЗНЕС | **1.путешествие по путеводителю** – тестирование по мануалу  **\*путешествие блоггера** – необходимо поискать, что пишут про подобные приложения пользователи  **\*путешествие недовольного всезнайки** – все негативные отзывы  **2.денежное путешествие** – тестирование тех областей приложения, которое приносит деньги  **\*путешествие скептика** – тестирование непопулярных способов (оплаты) достижения цели  **3.потушествие по ориентирам** – тестирование заранее оговоренных областей приложения, вне зависимости от приоритета  **4.путешествие интеллектуала** – задача тестировщика – задавать сложные вопросы приложению  **\*путешествие заносчивого американца** – когда создаем какие-то записи в приложении со специфическими данными.  **5.потешествие почты FedEx –** проверяем жизнь данных в системе (где хранятся, где создаются и т.д.)  **6.путешествие после работы** – тестирование, при котором проверяются различные задачи, выполняемые на сервере вне рабочее время приложения.  **7.путешествие мусорщика** – самые низкоприоритетные фичи / области приложения |
|  |
| ИСТОРИЧЕСКИЙ | **1.путешествие с «плохим соседом»**  **2.путешествие по музеям –** тестирование областей приложение, в которые месяц и больше тестировщики не заходили  **3.путешествие по уже посещенным местам –** тестирование тех областей, которые работали в предыдущих версиях |
| ТУРИСТИЧЕСКИЙ | **1.путешествие суфлера** – тестирование альтернативных способов выполнения действий  **2.путешествие по переулкам** – околоважные вещи (не приоритетные, не приносят никакой пользы бизнесу)  **3.путешествие «по ночам»** – оставляем приложение запущенным на длительный период времени (на ночь) |
| РАЗВЛЕКАТЕЛЬНЫЙ | **1.путешествие коллекционера** – создание всех записей, которые только возможны в системе (заполнить адрес – заполняем и т.д.)  **2.путешествие одинокого бизнесмена –** тестирование самого длинного пути выполнения каких-то действий  **3.путешествие супермодели** – непосредственно тестирование интерфейса приложения  **4.test one get one free** – тестирование изменений одной и той же записи из разных окон браузера или из разных девайсов (вк открываем на телефоне и браузере, что-то делаем и там, и там, и пытаемся сохранить одновременно)  **5. путешествие по шотландским пабам** – тестирование таких модулей и компонентов приложения, про которые мало кто знает |
| ОТЕЛЯ | **1.тур испорченный из-за дождя (испорченное путешествие)** – тестирование отмены всех действий, возможных в системе (начинаем логинится, вводим логин пароль и нажимаем отмену)  Отмена не работает как сохранение!!!  **2.путешествие ленивца** – тестирование самых кратких проверок, что записи создаются с минимальными данными (только обязательными полями) => если разрешен логин с 1 символов – проверяем |
| ПЛОХОЙ | **1.путешествие саботажника** – все негативные невалидные значения  **2.антисоциальное путешествие –** хакерские атаки  **3.путешествие человека, склонного к повторяющимся действия** – повторяющиеся действия  (на ссылку нажать 10 раз) |

# ЛК – 7. ТЕСТИРОВАНИЕ ТРЕБОВАНИЙ

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

Качественное приложение должно делать только то, что нужно и полезно для пользователя.

## Важность требований:

* позволяют понять, что и с соблюдением каких условий система должна делать
* предоставляют возможность оценить масштаб изменений и управлять изменениями
* являются основой для формирования плана проекта (в том числе плана тестир-ния)
* помогают предотвращать или разрешать конфликтные ситуации
* упрощают расстановку приоритетов в наборе задач. Позволяют объективно оценить степень прогресса в разработке проекта.

График

Чем раньше обнаружим ошибки – тем дешевле их исправить (позже – дороже)

**Формализация требований** – любая форма фиксации требования.

Является валидной на проекте при условии, что все члены команды договорились о ее приемлемости.

## Как могут быть оформлены требования:

* спецификация
* техническое задание
* варианты использования (как произведение, читая use case мы понимаем как будет работать приложение)
* user story (перечисляется тем, какие возможности есть у пользователя)
* mockups/wireframes (понимаем расположение элементов и какие они будут; последовательность, куда и как кликаем; но описания нету)
* дизайн (видим явное расположение эл-тов; есть с чем сравнить)

## Уровни и виды требований:

* бизнес требования
* пользовательские требования
* бизнес-правила
* функциональные требования
* нефункциональные требования
* требования к интерфейсам
* требования к данным

## Бизнес требования

Выражают цель, ради которой разрабатывает продукт.

Результатом выявления требований на этом уровне является общее видение – документ, который как правило, представлен простым текстом и таблицами. Нет детализации поведения системы и иных технических характеристик, но могут быть определены приоритеты решаемых бизнес-задач, риски и т.д.

Используются для проведения валидации и приемочного тестирования.

*Пример:*

* нужен инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты
* необходимо в 2-3 раза повысить количество заявок, обрабат.1 оператором за смену
* нужно автоматизировать процесс выписки товарно-транспортных накладных на основе договоров

## Пользовательские требования

Описывают задачу, которые пользователь может выполнять с помощью разрабатываемой системы (реакцию системы на д-вия пользователя, сценарии работы пользователя). Т.к. появляется описание поведения системы, требования этого уровня мб использованы для оценки объема работ, стоимости проекта, времени разработки и т.д.

Используются на системном уровне.

* Варианты использования
* Пользовательских историй
* Пользовательских сценариев

## Бизнес-правила

Описывают особенности принятых в предметной области (и/или непосредственно у заказчика) процессов, ограничений и иных правил. Эти правила могут относится к бизнес-процессам, правилам работы сотрудников, нюансом работы ПО и т.д.

## 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.

Часто используются на компонентном/интеграционном уровнях тестирования. Используются для видов функционального тестирования.

* В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе
* Система должна автоматически выполнять резервное копирование данных ежедневно в указанный момент времени
* Электронный адрес пользователя, вводимый при регистрации, должен быть проверен на соответствие требованиям RFC822.

## Нефункциональные требования

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

Используются на компонентном, интеграционном и системном уровнях. Используются для видов нефункционального тестирования.

* При одновременной непрерывной работе в системе 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов
* Ни при каких условиях общий объем используемой памяти не должен превышать 2 ГБ
* Размер шрифта для любой надписи на экране должен поддерживать настройку в диапазоне от 5 до 15 пунктов.

## Требования к интерфейсам

Описывают особенности взаимод.разрабатываемой сист. с др.системами и ОС.

* Обмен данными между клиентской и серверной частями приложения при осуществлении фоновых AJAX-запросов должен быть реализован в формате JSON
* Протоколирование событий должно вестись в журнале событий ОС
* Соединение с почтовым сервером должно выполняться согласно RFC3207 («SMTP over TLS»)

## Требования к данным

Описывают структуры данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой системы. Часто сюда относят описание БД и особенностей её использования.

* Все данные системы, за исключение пользовательских документов, должны хранится в БД под управлением СУБД MySQL
* Пользовательские документы должны хранится в БД под управлением СУБД MongoDB.
* Информация о кассовых транзакциях за текущий месяц должна хранится в операционной таблице, а по завершении месяца переносится в архивную.

## Баги в требованиях:

* Противоречия – написано одно, делается другое
* Неоднозначность требования – система должна быть userfriendly
* Неполное требование – система не должна давать пользователю ввести неправильный login/password
* Отсутствие единого стиля в работе ПО (на 3 вкладках есть подтверждение удаления, а на 4 нет)
* Невозможность реализовать требования
* Отсутствие трассируемости – требования прослеживаются по всем документам (кнопка есть, называется одинаково и выполняется аналогично)

Баги в требованиях оформляются в виде вопросов и по возможности уточняются.

## Способы уточнения требований:

* интервью;
* анкетирование;
* наблюдения за фокус-группой;
* анализ похожих систем;
* мозговой штурм;

## Оформление вопросов

* вопросы пишите короткими и простыми;

(ответ либо да, либо нет)

* вопрос не должен содержать «или»;

(непонятно на какую часть вопроса отвечает заказчик)

* формулируйте вопрос так, чтобы ответ на него был максимально коротким;
* обдумывайте ответ, который можно получить;
* предлагая улучшения, подкрепляйте их фактами и весомыми доводами;

(привести пример)

* просьбы подкрепляйте вежливыми оборотами;

(please)

## Примеры

|  |  |  |
| --- | --- | --- |
| **Требование** | **«плохие» вопросы** | **Хорошие вопросы** |
| Приложение должно быстро запускаться | \*насколько быстро?  \*а если не получится?  \*всегда ли? | \*каково максимально допустимое время запуска приложения?  \*допускается ли фоновая загрузка отдельных моментов?  \*что является критерием того, что приложение закончило запуск? |
| неоднозначность |
| Опционально должен поддерживаться экспорт документов в pdf | \*любых документов?  \*в pdf какой версии должен производится экспорт?  \*зачем!? | \*при каких условиях будет доступен экспорт в PDF?  \*для каких документов будет поддерживаться экспорт?  \*какие действия необходимо выполнить, чтобы экспортировать документ в PDF? |
| неоднозначное, неполное |
| Если дата события не указана, она выбивается автоматически | \*а если указана?  \*а если дату невозможно выбрать автоматически?  \*а если даты у события нет? | \*возможно, имелось в виду, что дата генерируется автоматически, а не выбирается?  Если «да», то по какому алгоритму она генерируется?  Если «нет», то из какого набора выбирается дата и как генерируется это набор? |
| неоднозначное, неполное |

Ограничение в 30 символов:

\*запретить вводить 30

\*дать ввести сколько хочет, а когда нажмет на копку, система не даст ввести такое количество

!все кнопки описаны

!кнопки описаны полностью!

?а что будет если я введу больше, не введу

!проверять на форматы

!задавать больше вопросов

# ЛК – 8. НЕФУНКЦИОНАЛЬНЫЕ ВИДЫ ТЕСТИРОВАНИЯ

**Нефункциональное тестирование** – тестирование, которое проводится для оценки характеристик ПО. Проверяется не корректность работы функций приложения, а сопутствующие характеристики.

## Виды нефункционального тестирования:

\* тестирование производительности

\* тестирование безопасности

\* тестирование эргономичности (usability testing)

\* тестирование совместимости (с браузерами, платформами)

\* 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)

## Тестирование производительности

это комплекс тестов определяющих поведение приложения при нормальной и предельной нагрузках.

**Подвиды нагрузочного тестирования**

### тестирование производительности

тестирование, которое проводится с целью определения, как быстро работает вычислительная система или ее часть под определенной нагрузкой (меряется на стороне сервера)

Пример: скорость генерации отчета по определенным параметрам из фильтра.

### стресс-тестирование

тестирование проводится для определения надежности системы во время экстремальных и диспропорциональных нагрузок и отвечает на вопросы о достаточной производительности системы в случае, если текущая нагрузка сильно превысит ожидаемый результат.

К экстремальным нагрузкам относится резкое почти единовременное повышение нагрузки до максимальных величин.

Пример: начало трансляции футбольного матча.

Задача – какое количество пользователей система поддерживает.

Нагрузка идет вертикально.

### нагрузочное тестирование

тестирование, которое проводится для того, чтобы оценить работоспособность приложения под заданной ожидаемой нагрузкой.

Этой нагрузкой мб, например, ожидаемое количество одновременно работающих пользователей приложения, совершающих заданное число итераций за интервал времени.

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

Отличие от тестирования производительности в параметрах.

В нагрузочном тестировании нагрузка идет ступенчато, т.е. пользователи добавляются постепенно (сначала 10, +10, +10 и так м.добавлять до 1000 и т.д.).

Задача – посмотреть с какого момента (количества пользователей) система начинает деградировать.

#### Процесс нагрузочного тестирования

1. анализ требований к ПО
   1. ожидаемое кол-во одновременно работающих пользователей
   2. ожидаемое время отклика системы при максимальных нагрузках
   3. ожидаемое состояние системы при максимальных нагрузках
   4. ожидаемое поведение системы при максимальных нагрузках
2. определение основных сценариев использования приложения (по этим сценариям будут бегать виртуальные пользователи)
3. запись и отладка скриптов (сценариев)
4. запуск скриптов с необходимым количеством пользователей + снятие показателей
5. анализ причин возникших проблем в приложении + подготовка возможных решений
6. внедрение решений + повторный запуск тестов ( повторяются пока не решатся проблемы с производительностью)

### тестирование стабильности

проводится с целью убедится в том, что приложение выдерживает ожидаемую нагрузку в течении длительного времени.

При проведении этого вида тестирования осуществляется наблюдений 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.

Нагрузка должна быть в течении всего времени максимальная.

### тестирование на больших объемах данных

тестирование производительности системы на больших объемах данных

## Тестирование безопасности

комплекс тестов определяющих степень уязвимости ПО к различным атакам

## Тестирование эргономичности

исследование, выполняемое с целью определения, удобен ли некоторый искусственный объект (веб-страница, пользовательский интерфейс или устройство) для его предполагаемого изменения. Это метод оценки удобства продукта в использовании, основанный на привлечении пользователей в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

Пользователи должны входить в целевую аудиторию приложения.

Если тестировщик не входит в целевую аудитория – все улучшения эргономичности не считаются значимыми.

## Тестирование совместимости

тестирование, целью которого является проверка корректной работы приложения в определенном окружении (как все стоит на своих местах)

\*аппаратная платформа

\*сетевые устройства

\*периферия (принтеры, CD/DVD-приводы, веб-камеры и т.д.)\

\*ОС (Unix, Windows, MacOS..)

\*БД (Oracle, MS SQL, MySQL..)

\*Системное ПО (веб-сервер, файрволл, антивирус…)

\*браузеры (Internet Explorer, Firefox, Opera, Chrome, Safari)

## UI тестирование

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

Отличие от тестирования совместимости – там проверяем, как стоят кнопки, на своих ли местах; а в UI тестировании проверяем цвета, корректность используемого шрифта, тип и размер шрифта, корректное применение стилей, margin, padding, т.е. все, на что не влияет браузер.

## Тестирование глобализации

\*наличие 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

### локализация ПО

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

Задаем главный вопрос «все ли элементы страницы переведены?»

Баг – \*название страницы не переводится, placeholder не перевелся;

\*текст на картинках тоже должен переводится (реклама в картинке)

\*особенности (для евреев не показывать свинину и т.д.)

\*ОАЭ (текст справа налево, картинки женщин и детей и т.д.)

### интернационализация ПО

технологические приемы разработки, упрощающие адаптацию продукта к языковым и культурным особенностям региона (регионов), отличного от того, в котором разрабатывался продукт

Предполагает специфический формат хранения данных, универсальных для поддерживаемых регионов.

К данным **относится**:

\*дата (американский формат (месяц, день, год);

английский формат (день, месяц, год));

\*время(am/pm; 24ч),

\*разделители в числах,

\*разделители в суммах валют и т.д.

### отличие локализации от интернационализации

интернационализация проводится на начальных этапах разработки, а локализация – для каждого целевого языка.

## 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 ПО, 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 на соответствие 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 ПО на корректность обработки прерываний в работы приложения

\*входящие звонки

\*смс сообщения

\*системные сообщения (низкий уровень зарядки батареи)

\*screen-lock

\*уход в спящий режим

## Тестирование соединения.

тестирование мобильного ПО на корректность работы на различных типах соединения

\*2G

\*3G

\*4G

\*Wi-Fi

\*EDGE

\*no connection

# ЛК – 9. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ

**Автоматизированное тестирование** – часть процесса тестирования на этапе контроля качество в процессе разработки ПО. Оно использует программные средства для выполнения тестов и проверки результатов выполнения, что помогает сократить время тестирования и упростить его процесс.

Относится к функциональному виду тестирования и проверяет корректность работы отдельных методов/ компонентов/всего приложения.

## Уровни авто-тестирования:

\* уровень модульного тестирования (unit tests)

\* уровень функционального тестирования (non-UI tests)

\* уровень тестирования через пользовательский интерфейс (UI tests)

## 1. Модульное тестирование

**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.

Пример:

class CalculatorTests{

public void sum\_2plus5\_7returned(){

//arrange

var calc = new calculator();

//act

var res = calc.sum(2,5);

//assert

assert.areequal (7, res)

}}

### Правила UNIT-тестов

Тесты должны быть:

\*достоверными

\*не зависеть от окружения, на котором они выполяются

\*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 загрудаться в системе контроля версий

\*названия должны быть «говорящими»

[тестируемый\_метод]\_[сценарий]\_[ожидаемое\_поведение]

class CalculatorTests{

public void sum\_2plus5\_7returned(){…}

\*единый стиль написания тела теста

(3 составляющие (arrange, act, assert)) – пример выше

**как не надо делать!**

class CalculatorTests{

public void sum\_2plus5\_7returned(){

Assert.AreEqual(7, new Calculator().sum(2,5));}

\*использование готовых фреймворков.

UNIT - тесты могут писаться после того, как написан сам код и в этом случае это будет контроль качества.

UNIT – тесты могут писаться перед написанием самого кода и в этом случае это будет обеспечение качества.

## 2. NON-UI тестирование

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

Пример:

package CountriesRestTests;

…..

public class GetTest{

@Test

public void getRequestFindCapital() throws JSONException{

//выполняем запрос get для доступа ко всем пар-рам ответа

Response resp = get(“http://restcountries.eu/..../name/belarus”);

JSONArray jsonResponse = new JSONArray(resp.asString());

//получение параметра capital (столицы беларуси)

String capital = jsonResponse.getJSONObject(0).getString(“capital”);

//проверка, что столицей является минск

AssertJUnit.assertEquals(capital, “Minsk”);

}}

**Инструменты** – SoapUI, RestSharp, postman, rest-assured;

## 3. UI тестирование

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

Инструмент -> Браузер -> Сервер

Во время данного тестирования тест полностью имитирует работу пользователя (открывается браузер, нажимаются кнопки, вводятся данные). При этом, если элемент не догрузился или отсутствует на странице, то действие не мб выполнено. Это делается специально, для того, чтобы предотвратить ошибки когда у реального пользователя есть баг, а тест его не находит.

**Инструменты** – VisualStudio, Se, QuickTestProfessional, Ranorex, TestStudio, appium, Protractor, Calaba.sh

### Подходы в UI автоматизации

Driven означает, что тесты зависят от каких-то определенных обстоятельства

\* data-driven подход (данные)

\* keyword-driven подход (ключевые слова)

\* behavior-driven подход (поведение)

**Data-driven подход**

Data-driven подход используется в приложениях, внешний вид и работа которых во многом зависит от введенных данных

(калькулятор)

**Keyword-driven подход**

Часто используется, когда команда автоматизации готовит карту объекта, а команда ручных разработчиков составляет тест так, как им нужно**.** Тест строится с помощью набора объектов.

Планирование + Реализация

\*создание карт объекта реализация фреймворка,

\*создание набора ключ.слов обеспеч.ключ слова

|  |  |  |
| --- | --- | --- |
| Объект | Действие | Данные |
| LoginPage | Open |  |
| InputLogin | Enter text | User |
| InputPassword | Enter text | password |
| ButtonLogin | Click |  |
| HomePage | assertLoggedIn |  |

**Достоинства и недостатки подхода**

+: распределение задач по написанию тестов

+: понимание нетехническими членами команды, что происходит в тесте

-: удлинение времени создания тестов

-: сложно поддерживать тесты, когда их >300 -> сложно избежать дублирования объектов

**Behavior-driven подход**

В подходе, который зависит от поведения системы активно используются ключевые слова, которые объединяются в глоссарии, где прописывается что выполняется в тесте, какое действие выполняется в тесте под каждым словом.

Тесты всегда описываются по структуре, используя слова

* given (дано),
* when (что делаем),
* then(результат (что получаем после проведенных действий))

Данный подход распространяется не только на автоматизацию, но и на весь проект в целом. Все требования оформляются в виде подобных сценариев (given, when, then).

Мануальные тестировщики на таком проекте полностью отсутствуют и все тестирование проводится посредством автоматизации.

**Достоинства и недостатки подхода**

+: сокращается время на написание предварительно тестовой документации, т.к.сами бизнес-аналитики сами пишут сценарий

+:нетехнические члены команды могут легко понимать, что происходит в тесте

-: удлинение времени создания тестов

-: сложно поддерживать тесты, когда их >300 -> сложно избежать дублирования объектов

Язык Геркина (Gherkin)

## Области для автоматизации:

\*труднодоступные места в системе

\*часто исп-мая функциональность, риски от ошибок в которой достаточно высоки

\*рутинные операции

\*длинные end-to-end сценари

\*проверка данных, требующих точных мат.расчетов.

Пирамида автоматизации

(большая часть должна быть покрыта UNIT-тестами, дальше интеграционные тесты, меньше всего дб UI-тестов)

## Процесс автоматизации тестирования

\*подготовка и планирование

\*дизайн и разработка

\*прогон тестов и сопровождение

### 1.Подготовка и планирование

* анализ требований клиента к автоматизации
* анализ тесткейсов для ручного тестирования
* оценка пригодности приложения к автоматизации
* выбор инструмента автоматизации
  + выбор системы автозапуска тестов
  + выбор инструмента отчетности
* создание пилотных скриптов
* установка тестового окружения

### 2.Дизайн и разработка

* дизайн архитектуры (модули, фреймворки)
* подготовка тестовых данных (если будут использоваться)
* определение повторно используемого кода и реализация многократно используемых функций, библиотек и фреймворков
* разработка и отладка тестовых скриптов

### 3.Прогон тестов и сопровождение

* запуск автоматических скриптов
* анализ результатов запуска

(Выбираем все тесты, которые упали в процессе прогона и запускаем их еще раз, таким образом отбираем только те, которые упали второй раз.

У тестов, которые упали второй раз тестировщик проходит шаги вручную или просматривает log. Задача тестировщика – понять где баг (в тесте или прилож))

* создание отчетов о результатах запуска
* поддержание тестов в актуальном состоянии

# ЛК – 10. ТЕСТИРОВАНИЕ БЕЗОПАСНОСТИ

**Тестирование безопасности** - комплекс тестов определяющих степень уязвимости ПО к различным атакам

**Уязвимость** – недостаток или слабость проектирования, реализации или эксплуатации системы, который мб 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, таким как данные в БД или в файлах системы), используя уязвимость.

Open Web Application Security Project ([www.owasr.org](http://www.owasr.org)) – об уязвимостях для приложений (веб и мобильных)

## Компоненты эффективного процесса тестирования:

* люди – убедится, что люди обучены и проинструктированы
* процесс – убедится, что есть адекватные стандарты и стратегии обеспечения качества
* технология – убедится, что процесс показал себя, как эффективный в процессе его внедрения и выполнения.

## Техники тестирования безопасности:

* ручное инспектирование и ревью;
* моделирование угроз;

(продумыв. какие угрозы мб у прилож, а также модель нарушителя)

* ревью кода;
* тестирование на проникновение;

(самый неуспешный вид тестирования)

## Эффективный процесс тестирования:

* перед началом разработки
  + определения жизненного цикла разработки ПО;

(каскадная модель)

* + 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
  + детальный код ревью
* во время деплоймента
  + тестирование на проникновения
  + тестирование конфигурации приложения
* поддержка приложения
  + периодическое ревью процесса
  + периодическое тестирование на проникновения

## Фазы тестирования безопасности:

### пассивная фаза

\* сбор информации

\* изучение приложения

### активная фаза

\*сбор информации

\*тестирование конфигурации и Deployment Management’a

\*тестирование ролей и прав доступа

\*тестирование аутентификации

\*тестирование авторизации

\*тестирование состояния сессий

\*тестирование входных данных

\*обработка ошибок

\*криптография

\*тестирование бизнес логики

\*тестирование клиентской части ПО

#### 1. Сбор информации

* Поиск утечки информации с помощью поисковиком

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

* Определение веб-сервера

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

* Ревью метафайлов веб-сервера

**Цель:** найти утечку 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 мб уязвимы.

* Ревью комментариев и мета-данных

**Цель:** найти утечки информации в комментариях кода и мета-данных

* Идентификация точек входа

**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
  + Firewalls;
  + Load balancers;
  + etc.

#### 2. Тестирование конфигурации

* Тестирование сети/инфраструктуры
  + тестирование известных багов веб-серверов и серверов приложений
  + тестирование административных инструментов (которые участвуют в работе приложения)
* Тестирование платформы приложения
  + хранение сенситивных д-х в системных файлах, логах и т.п. файлах.
* Тестирование содержания хранимых файлов
  + Содержимое .asa и .inc файлов
  + Содержимое txt файлов, .bak, .old и других файлов на сервере
* Тестирование HTTP методов
  + PUT: позволяет загрузить файлы(вредоносные) на сервер
  + DELETE:мб исп-н для удаления важных для работы файлов
  + CONNECT: мог позволить клиенту использовать сервер как прокси
  + TRACE: отражает клиенту то, что было отправлено на сервер

#### 3. Тестирование ролей и прав

* Тестирование ролей

**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.

#### 4.Тестирование аутентификации

* Тестирование передачи логина/пароля
  + Отправка д-х используя POST запрос через HTTP протокол
  + Отправка д-х используя POST запрос через HTTPS протокол
  + Отправка д-х используя POST запрос через HTTPS протокол, но используя HTTP протокол
  + Отправка д-х используя GET запрос через HTTPS протокол
* Тестирование популярных логина/пароля
  + Admin/admin;
  + Admin/passw0rd (p@ssword)
* Тестирование путем подстановки URL

<http://www.site.com/page.asp?authenticated=no>

<http://www.site.com/page.asp?authenticated=yes>

# ЛК – 11. ТЕСТОВАЯ ДОКУМЕНТАЦИЯ

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

Противовес тестовой документации – **проектная документация**: функциональные / нефункциональные / пользовательские и др. требования, проектный график, проектный план, дедлайны и правила проекта.

Цель проектной документации – упорядочить и облегчить работу на проекте.

Тестовая документация делится на **2 части**:

|  |  |
| --- | --- |
| Тестовая документация | |
| Планирование тестирования | Отчетность |
| * + тест план   + график тестирования   + матрица устройств   + матрица прослеживаемости   + тестовый набор   + тест сценарии   + текст кейсы   + чеклист | * + отчеты об ошибках   + отчеты о результатах тестирования |

## Тест план

**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 разрешения.

**Цели написания тест плана:**

* продумать стратегию тестирования ПО
* 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

Под **стратегией тестирования** понимаются:

\*виды тестирования, которые будут проводится над фичами и приложением

\*приоритеты (что будет тестироваться в первую очередь и т.д.)

\*тестовая документация и ее соотношение с видами тестирования

\*браузеры или поддерживаемые устройства, на которых будем тестировать

\*инструменты, которыми будем пользоваться во время тестирования

\*потенциальные проблемы, с которыми можем столкнуться в процессе тестирования и план действий, если столкнемся

\*мб прописаны обязанности каждого члена команды тестирования

## График тестирования

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

**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

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

Матрица устройств составляется для тестирования совместимости.

**Цель создания матрица устройств:**

* определить оптимальное количество устройств, необходимых для тестирования;
* убедится, что устройства с различными версиями необходимого оборудования участвуют в тестировании;
* сосредоточить усилия тестировщика на устройствах, оптимальное полно покрывающих требования проекта;
* обеспечить прозрачность тестирования для заказчика.

## Матрица прослеживаемости

**Матрица прослеживаемости** – документ, используемый для определения покрытия требований тестовой документации.

**Цель создания матрицы прослеживаемости:**

* обеспечить должное покрытие всех функциональных и нефункциональных требований тестами (убедиться, что ничего не упустили)

## Тестовый набор

**Тестовый набор** – документ, вмещающий в себя тестов/тестовых случаев/сценариев, собранных с определенной целью.

**Цель создания тестового набора:**

* сгруппировать тестовые случаи по:
  + видам тестирования;
  + уровням тестирования;
  + приоритету (smoke, critical path, extended (расширенный))
* облегчить распределение объема тестирования в команде;
* облегчить оценку трудозатрат тестировщика.

У каждой из проверок ставится либо #, либо label.

## Тест сценарий

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

**Виды тестирования,** для которых пишут сценарии:

\*исследовательское тестирование

\*end-to-end тестирование

\*role-based тестирование

## Тест кейс

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

**Виды тестирования,** для которых пишут тест кейсы:

\*позитивное тестирование

\*негативное тестирование

\*CRUD тестирование

Тестовый сценарий – длинный (от 30 до 60 шагов) и имитирует работу реального пользователя. Тест кейс предназначен для тестирования одного требования. Для того, чтобы протестировать работоспособность копки «Cancel» достаточно нажать на нее и проверить результат с ожидаемым результатом. Ожидаемый результат должен быть описан в тест кейсе, а в сценариях описываем только шаги.

## Чеклист

**Чеклист** – документ, перечисляющий идеи для проверок. 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

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

Для чего пишут отчеты об ошибках:

\*для истории

\*для прозрачности

\*чтобы вся команда была в курсе

\*для порядка (если кто-то забудет(тестировщик/разработчик))

## Отчет о результатах тестирования

**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.

## Баги

**Жизненный цикл** бага (м.почитать в интернете)

1) открыт (open) – означает, что он есть в системе, и он неисправен.

2) в процессе (in progress) - разработчик взял баг в исправление

3) решенный (resolved)

**Разновидности резолюций** от разработчика:

\*исправленный (fixed) – баг был исправлен

\*cannot reproduced – разработчик не смог воспроизвести баг по нашим шагам

\*Wont-fix - баг не в приложении, а в фреймворке

\*Differed (отложенный) - для исправления необх. переписать приложение

\*not a bug - это не баг, это фича

Если все хорошо:

4.1) закрыт (closed) + пишем коммент, что он исправлен и т.д.

Не м.воспроизвести:

4.2) Reopen – переоткрываем баг (см.п.1 – все начинается сначала)

# ЛК – 12. ТЕСТ ДИЗАЙН

**Тест дизайн –** 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.

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

**Требование:** Login field, unique = true;

* тестовый случай: проверить уникальность поля login
* тестовые условия: если значение в поле login не уникально, показать сообщение
* тестовые данные: в БД существует пользователь с login = value.

## Категории техник ТД:

* разработка тестов методом черного ящика
  + для описания задач, которые дб 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 от том, как спроектировано ПО (например, на основе программного кода и подробного описания проектного решения):
  + для ПО мб измерена величина покрытия для имеющихся тестовых сценариев, и последующие тестовые сценарии могут разрабатываться для систематического увеличения покрытия
* метод создания тестов на основе опыта
  + для определенных 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

|  |  |
| --- | --- |
| **Метод на основе опыта** | |
| **Метод черного ящика** | **Метод белого ящика** |
| Техника базируется на анализе спецификаций или других проектных документов, описывающих требования к системе. | Техника базируется на анализе внутренней структуры компонентов системы. |

## ТД МЕТОДОМ ЧЕРНОГО ЯЩИКА

* эквивалентное разбиение
* анализ граничных значений
* тестирование таблицы решений
* тестирование таблицы переходов
* тестирование по сценариям использования

### 1. Эквивалентное разбиение

Входные данные для ПО или системы разбиваются на **группы,** от которых ожидается сходное поведение, то есть они должны обрабатываться аналогичным образом. Данные группы называются **классами эквивалентности.**

**Пример:**

|  |  |  |
| --- | --- | --- |
| **Система подбора кредита** | | **Класс эквивалентности** |
| Возраст до 18 лет | Кредит не разрешен | #1: от 1 до 17 |
| Возраст от 18 до 65 лет | \*потребительский кредит  \*кредит на недвижимость  \*кредит на покупку авто | #2: от 18 до 65 |
| Возраст старше 65 лет | Потребительский кредит | #3: от 66 до 99 |

### 2. Анализ граничных значений

Поведение на границах эквивалентных областей имеет наибольшие шансы быть некорректным, таким образом границы являются потенциальным источником дефектов.

Минимальное и максимальное значения сегмента называются **граничными значениями.** (min-1, min, average, max, max+1)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Система подбора кредита** | | | | | | | | | | | | | |
| Возраст до 18 лет  Класс эквивалентности #1: от 1 до 17 | | | | | | | | Кредит не разрешен | | | | | |
| **Min -1:** | | | | | Пустое поле «возраст» | | | | Сообщение об ошибке | | | | |
| **Min:** | | | | | 1 | | | | «Кредит не разрешен» | | | | |
| **Average:** | | | | | 9 | | | | «Кредит не разрешен» | | | | |
| **Max:** | | | | | 17 | | | | «Кредит не разрешен» | | | | |
| **Max +1:** | | | | | 18 | | | | «Потребительский кредит  Кредит на недвижимость  Кредит на покупку авто» | | | | |
| Возраст старше 65 лет  Класс эквивалентности #3: от 66 до 99 | | | | | | | | Потребительский кредит | | | | | |
| **Min -1:** | | | | | 65 | | | | 3 кредита | | | | |
| **Min:** | | | | | 66 | | | | «Потребительский кредит» | | | | |
| **Average:** | | | | | 80 | | | | «Потребительский кредит» | | | | |
| **Max:** | | | | | 99 | | | | «Потребительский кредит» | | | | |
| **Max +1:** | | | | | 100 | | | | Система не позволяет ввести больше 2 символов | | | | |
| **Дата рождения** (формат: дд.мм.гггг) | | | | | | | | | | | | | |
|  | Январь, март, май, июль, август, октябрь, декабрь | | | | | | Февраль (висок.год) | | | Февраль (не висок.год) | | Апрель, июнь, сент., ноябрь | |
| **Min -1:** | | 00 | | | | | 00 | | | 00 | | 00 | |
| **Min:** | | 01 | | | | | 01 | | | 01 | | 01 | |
| **Average:** | | 15 | | | | | 15 | | | 15 | | 15 | |
| **Max:** | | 31 | | | | | 29 | | | 28 | | 30 | |
| **Max +1:** | | 32 | | | | | 30 | | | 29 | | 31 | |
| **E-MAIL** (формат: local-part(64)@hostname.domain(255)) | | | | | | | | | | | | | |
|  | | | local-part | | | @ | | hostname | | | dot | | domain |
| **Min -1:** | | | *empty* | | | *empty* | | 1 character | | | e*mpty* | | 1 character |
| **Min:** | | | 1 char. | | | @ | | 2 char. | | | . | | 2 char. |
| **Average:** | | | 30 diff.char. | | | @ | | 150 char. | | | . | | 6 char. |
| **Max:** | | | 64 | | | @ | | 243 char. | | | . | | 11 char. |
| **Max +1:** | | | 65 | | | @@ | | 244 char. | | | . | | 12 char. |
| **E-MAIL**(размер: 256 символов) | | | | | | | | | | | | | |
|  | | | | Size | | | Example | | | | | | |
| **Min -1:** | | | | 5 char | | | @a.by; aa.by; [a@.by](mailto:a@.by); a@aby; [a@a.by](mailto:a@a.by); a@aa.by | | | | | | |
| **Min:** | | | | 7 char | | | ……. | | | | | | |
| **Average:** | | | | 150 char | | | ……. | | | | | | |
| **Max:** | | | | 256 char | | | ……. | | | | | | |
| **Max +1:** | | | | 257 char | | | ……. | | | | | | |

### 3. 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 обычно является наличие хотя бы одного теста для каждой колонки, что обычно включает в себя покрытие всех комбинаций триггерных условий.

### 4. 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.

### 5. Тестирование по сценариям использования

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

**Cистема документооборота**

Создать новый документ =>

Временный документ temporary =>

Сохраненный документ saved =>

Просмотренный документ reviewed =>

Утвержденный документ approved =>

Выполненный документ processed =>

Заархивированный документ archived

|  |  |  |  |
| --- | --- | --- | --- |
| **Тек.состояние** | **Событие** | **Действие** | **След.состояние** |
| null | «создать новый документ» | create | temporary |
| null | «создать новый документ» | discard | null |
| temporary | заполнение документа | save | saved |
| temporary | заполнение документа | cancel | null |
| saved | простор документа | review | reviewed |
| saved | просмотр документа | reject | rejected to creator |
| reviewed | утверждение документа | approve | approved |
| reviewed | утверждение документа | reject | archived |
| approved | выполнение д-вий по доку | process | processed |
| processed | архивация дока | archive | archived |

## ТД МЕТОДОМ БЕЛОГО ЯЩИКА

* компонентный уровень:

структура компонента ПО, т.е. операторы, альтернативы, ветви или определенные пути

* интеграционный уровень

структура мю представлена деревом вызова(диаграмма, в которой модули вызывают другие модули)

* системный уровень

структура может представлять собой структуру меню, бизнес-процессов или же схему веб-страницы

**Техники тест дизайна:**

1. тестирование покрытия операторов
2. тестирования покрытия альтернатив
3. тестирование покрытия условий

### 1. Тестирование покрытия операторов

**Тестирование покрытия операторов** – техника тестирования, которая включает в себя выполнение всех операторов хотя бы единожды. Полученная метрика позволяет высчитать количество операторов, которые были выполнены и которые вообще есть в коде.

Prints(int a, int b){

int result = a+b;

if(result>0) print (“Positive”, result);

else print (“Negative”, result)

};

Scenario1:

If A = 3, B= 9

Statement Covarsge: 5/7=71%; --Positive

Scenario2:

If A = -3, B= -9

Statement

### 2.Тестирование покрытия альтернатив

**Тестирование покрытия альтернатив** – техника тестирования, которая связана с тестированием ветвей (branches) (например, вариантов «True» и «False» для оператора «IF»). В методе тестирования альтернатив тестовые сценарии создаются для выполнения определенных результатов альтернатив. Ветви исходят из точек альтернатив в программном коде и показывают передачу управления различным участкам кода.

**Покрытие альтернатив** – вид тестирования потока управления, так как оно описывает прохождение определенного потока через точки альтернативы. Покрытие альтернатив более строгое, чем покрытие операторов: 100% покрытие альтернатив обеспечивает 100% покрытие операторов, но не наоборот.

### 3.Тестирование покрытия условий

**Покрытие условий и покрытие множественных условий** – более высокий уровень покрытия структуры кода после покрытия альтернатив. Техника предполагает покрытие всех условий, которые могут влиять на результат принятий решения.

## Тест дизайн на основе опыта:

1. предположения об ошибках (error guessing technique)
2. исследовательское тестирование (exploratory resting)

### 1.Предположения об ошибках

Организованным подходом к предположению об ошибках является создание списка возможных дефектов, и разработка тестов для атаки этих дефектов. Данный подход называется **атакой**, или **атакой на недочеты**. Списки дефектов и отказов могут быть созданы на основе опыта, доступной информации о дефектах и отказах и общего представления о том, почему ПО может отказать.

# ЛК – 13. ТЕСТ КЕЙСЫ

**Тест кейс (тестовый случай)** – формально описанный алгоритм тестирования программы, специально созданный для определения возникновения в программе определенной ситуации, определенны входных данных.

**Цель написания** тестовых случаев:

* подготовится к тестированию (основная)
* детально описать шаги тестирования и ожидаемый результат
* задокументировать требования
* облегчить передачу знаний по проекту
* подготовится к автоматизации тестирования

## Атрибуты тестового случая:

* **Идентификатор (ID)**
  + Уникален среди всех доков (tc№)
* **Краткое название тестового случая (summary, title)**
  + уникален среди всех тест кейсов
  + состоять только из существительных
* **Цель тестового случая (goal, aim, description)**
  + описано что происходит (мы проверяем) и при каких условиях
  + 1 предложение, кот. начинается с «verify that»
* **Предусловия (precondition)**
  + это нумерованный список шагов, кот.необх.вып-ть, чтобы добраться до тестируемого компонента
  + ожидаемый результат прописывать не нужно
* **Шаги (steps)**
  + описываются только те действия, которые направлены на проверку выбранного требования
* **Ожидаемый результат (expected result)**
  + прописывается для каждого шага
  + дб подробным и однозначным (страница открывается быстро)
* Постусловия (postcondition)
  + действия, которые необходимо выполнить, чтобы вернуть систему в исходное состояние
* **Статус (status)**
  + есть всегда (4 типа статусов):
    - не пройдет (not run)
    - успешно пройден (passed) <= ож.рез-т=акт.рез-т
    - неудачно пройденный (failed(id бага)) <= акт.рез-т != ож.рез-т
    - заблокирован (blocked) <= не м.вып-ть шаг/предусловие
* Уровень (level)
  + выставляется в соответствии с ур.тестир., для кот. он предназначен (компонентный, интеграционный, системный, приемочный)
* Приоритет (priority)
  + высокий – должны пройти в 1 очередь (оплата картой)
  + средний – 2 очередь
  + низкий – 3 очередь (отзыв, лайк)
* Автор (author)
  + автоматически выставляется в test management системах
* Комментарии (comments, notes)
  + любые пометки

То, что жирным – обязательно.

(пример ui test case, negative test case, positive test case)

## Правила написания ТС

* 1 требование = 1 тестовый случай
* Количество шагов не больше 7
* В steps/expected result – только то, что относится к цели тест кейса
* Шаги, которые не относятся к цели тест кейса – либо Preconditions, либо Postconditions
* Минимум кликов – максимум результата
* ТК должны идти последовательно, составляя сценарий
* ТК должны быть разделены на смысловые части
* Как можно меньше ссылок на сторонние документы
* Без картинок

# ЛК – 14. ТЕСТОВЫЕ СЦЕНАРИИ

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

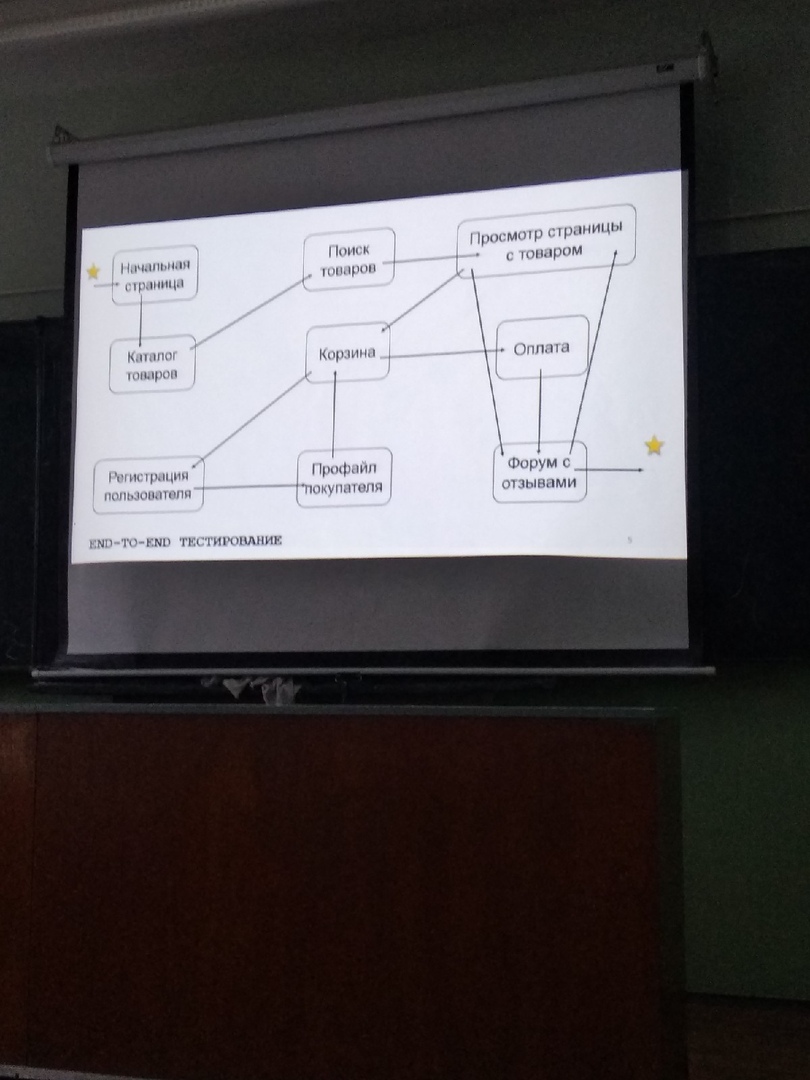
## Отличие тестового случая от тестового сценария:

|  |  |
| --- | --- |
| **Тестовый случай** | **Тестовый сценарий** |
| 1. проверяет одно требования  2.проверяет функциональность и внешний вид  3. содержит не больше 5-7 шагов  4. каждый шаг содержит ожидаемый результат | 1. проверяет много требований  2. проверяет только функциональность  3. «чем больше шагов, тем лучше»  4. ожидаемые результат не прописывается |

**Цель** написания тестового сценария:

Подготовится к проведению следующих видов тестирования:

* **end-to-end** – подход, при котором проводится сквозное тестирование системы по сценариям использования системы реальными пользователями



* **role-based testing** - тестирование, направленное на проверку ограничения прав доступа к конкретному функционалу на основе роди и привилегий пользователя;

|  |  |
| --- | --- |
| **Роль** | **Действия** |
| Content Manager | 1. заходит в admin часть приложения  2. добавляет новый товар в каталог  3. заполняет техническое описание товара  4. переводит товар в статус «новый» |
| Marketing Manager | 5. открывает товар со статусом «новый»  6. выставляет цену товара  7. создает систему скидок на товар  8. переводит в статус «готово» |
| Buyer | 9. открывает приложение и заходит под своим логином/паролем  10. на начальное странице видит новый товар  11. оформляет попытку  12. получает скидку согласно определенной выше системе  13. оплачивает товар  14. выходит из приложения |

* **exploratory testing** (исследовательское тестирования) - подход к тестированию, который подразумевает под собой одновременное изучение приложения, проектирование тестовых сценариев и их немедленное выполнение:
  + путешествие по путеводителю;
  + денежное путешествие;
  + путешествие по достопримечательностям;
  + путешествие по переулкам;
  + путешествие суфлера;
  + путешествие одного бизнесмена

**Отличие:**

end-to-end – 1 пользователь, много компонент

role-based – много пользователей, 1 компонент.

## Атрибуты тестового сценария:

* **идентификатор (id);**
* **краткое название (summary, title);**
* **описание сценария (description);**
  + 1-5 предложений, кот описывают какие цели, компоненты данного сценария
* **участники (primary actors);**
  + это роли, пользователи, которые фигурируют в сценарии
    - лаба (незарегистрированный, авторизованный и зарег)
    - (end-to-end – мин кол-во (1))
    - (role-based – множество пользователей)
    - система не участник
* **предусловия** **(precondition);**
  + действия, которые необходимо выполнять для того, чтобы подготовить систему к выполнению сценария.
* **основной сценарий** **(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
  1. пользователь просматривает товар
  2. пользователь добавляет товар в корзину
  3. пользователь выбирает оплату картой
  4. пользователь вводит данные карты
  5. пользователь подтверждает оплату
* **альтернативный сценарий (alternative flow)** – последовательность шагов или действий, ведущая к достижению цели пользователя, но которая подразумевает отклонение от направления по умолчанию
  1. пользователь просматривает товар
  2. пользователь добавляет товар в корзину
  3. пользователь выбирает оплату картой =>

*пользователь выбирает оплату WebMoney*

* 1. пользователь вводит данные карты
  2. пользователь подтверждает оплату
* **исключения (exceptional flow)** – последовательность шагов или действий, которая препятствует достижению цели, и которая подразумевает отклонение о правильной работы приложения
  1. пользователь просматривает товар
  2. пользователь добавляет товар в корзину
  3. пользователь выбирает оплату картой
  4. пользователь вводит данные карты =>

*пользователь вводит данные просроченной карты*

* 1. пользователь подтверждает оплату
* **статут** **(status);**
  + такой же как у тест-тексов
  + failed – если в процессе прохождения найдены баги, где поведение системы не соответствует требованиям или здравому смыслу.
* **приоритет** **(priority);**
  + как у тест-кейсов (высокий, средний, низкий)
  + указывает очередность выполнения сценариев
    - высокий – базовые действия пользователя (зафрендить)
    - средний – сценарии, которые затрагивают второстепенные функции (постить фотки в соц.сетях)
    - низкий – самые незначительные в приложении функции (поставить лайк, написать комментарий);

# ЛК – 15. ОТЧЕТЫ О РЕЗУЛЬТАТАХ ТЕСТИРОВАНИЯ

**Отчет о результатах тестирования** – документ, обобщающий результаты работ по тестированию и содержащий информацию, достаточную для соотнесения текущей ситуации с тест-планом и принятия необходимых управленческих решений.

## Цель написания отчетов:

* предоставить статистику по количеству проверок и результатам 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;

## Структура отчета:

* + 1. уровень/вид проведенного тестирования
    2. количество пройденных проверок
       - * общее количество
         * количество успешно пройденных
         * количество неудачно пройденных
         * количество заблокированных проверок
    3. количество найденных багов:
       - общее количество багов
       - количество critical багов
       - количество major багов
       - количество minor багов
       - количество trivial багов
    4. области/компоненты, которые наиболее повреждены багами (более 75% проверок failed)
    5. проблемы, с которыми столкнулись в процессе выполнения задания

В отчеты также могут быть указаны номера билдов, на которых проводилось тестирование (в 1 пункте), а также могут перечисляться члены команды, которые проводили данные виды тестирования.

# ЛК – 16. ТЕСТ ПЛАН (тест стратегия)

**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 разрешения **(лк 11)**

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

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

Стратегия зависит от цели: что ожидается от тестировщиков. И уже в зависимости от цели выстраивается стратегия тестирования.

## Виды тест стратегий:

* аналитические стратегии – требуют анализа для того, чтобы сфокусировать тестирование и ограничить по времени и усилиям.
  + примеры:
  + тестирование, основанное на рисках
    - тестирование, основанное на требованиях
* стратегии, основанные на моделях
  + примеры:
  + тестирование, основание на моделях использования приложения
* методические стратегии – ориентированы на результат
  + 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 др
* реактивные стратегии
  + примеры:
  + исследовательское тестирование
* консультативные стратегии
  + примеры:
  + тестирование, основано на сценариях и данных, предоставленных заказчиком ПО.
* стратегии, исключающие регрессионное (повторное) тестирование
  + пример:
  + 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 конца тестирования
* метрики, собираемые в процессе тестирования
* инструменты, используемые в тестирования
* окружение, где проходит тестирование
* процесс контроля качества и его метрики
* дефект менеджмент
* 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
* график тестирования и бюджет
* циклы тестирования и их соотношение с реальным планом проекта
* взаимоотношения между командой разработки и тестирования, результаты работы
* 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

**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 тест план.

# ЛК – 17. ОТЧЕТЫ ОБ ОШИБКАХ

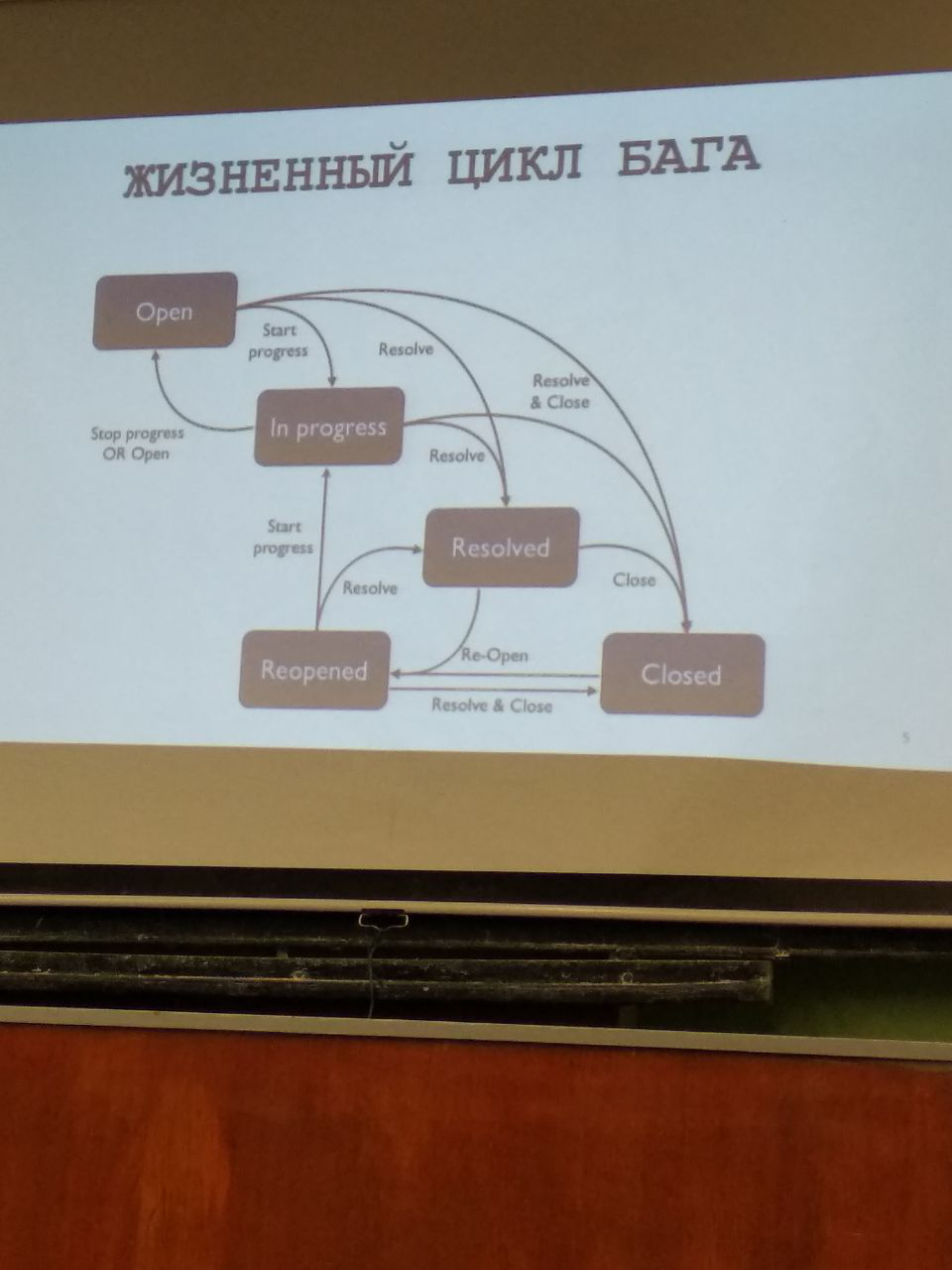
**Отчеты об ошибках** – документ, описывающий и приоритизирующий обнаруженный дефект, а также содействующий его устранению.

**Дефект –** любое отклонение фактического результата от ожиданий наблюдателя, пользователя, сформированных на основе требований, спецификаций, иной документации или опыта и здравого смысла.

## Цель отчета об ошибке:

* предоставить информацию о проблеме:
  + уведомить проектную команду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы
* приоритизировать проблему:
  + определить степень опасности и желаемые сроки ее устранения
* содействовать устранению проблемы:
  + предоставить необходимые подробности для понимания сути случившегося, а также анализ причин возникновения проблемы и рекомендации по исправлению ситуации

## ЖЦ бага:



Разработчик не имеет права закрывать баг.

## Разновидности резолюций от разработчика:

\*исправленный (fixed) – баг был исправлен

=> проверить, что он исправлен (ожидаемый = актуальный);

\*cannot reproduced – разработчик не смог воспроизвести баг по нашим шагам

=> проверить окружение и шаги; мб записать видео + приложить логи;

\*wont-fix - не будет исправлен

=> баг не в приложении, а в фреймворке;

\*as designed – работает как задумано

\*differed (отложенный) - для исправления необх. переписать приложение

\*duplicate – дубликат (такой баг уже есть в системе)

=> найти дубликат, написать коммент;

**Атрибуты отчета об ошибке:**

* идентификатор (id)
* краткое описание (summary, title)
* подробное описание (description)
  + шаги для воспроизведения (steps to reproduce)
  + актуальный результат (actual result)
  + ожидаемый результат (expected result)
* воспроизводимость (repeatability)
* важность (severity)
* срочность (priority)
* возможность обойти (workaround)
* комментарий (comment, note)
* среда для воспроизведения бага (environment)
* компонент (component)
* тег (tag, label)
* ответственный (assignee)
* приложение к багу (attachment)
* срок исправления бага (due date)
* версия билда, где был обнаружен баг (affect version)
* версия, в которой дб исправлен баг (fix version)
* создатель отчета об ошибке(reporter)
* оценка трудозатрат на исправление бага (original estimate)

|  |  |
| --- | --- |
| **Важность бага** | **Приоритет бага** |
| \*блокирующий (blocked)  \*критичный (critical)  \*значительный (major)  \*незначительный (minor)  \*тривиальный (trivial) | \*высокий (high)  \*средний (medium)  \*низкий (low) |

## Свойства качественных отчетов:

* тщательное заполнение всех полей точной и корректной информацией;
* отсутствие «размытых» формулировок («иногда неправильно работает», «UI сломан», «кнопка Export не работает»);
* вся информация, особенно ожидаемый результат, корректна;
* баг заведен на функциональность, которая разработана (неразработанный функционал за ошибку не считается)
* отсутствует жаргонная лексика («фигово подцепились чартники»)
* отсутствие критики кого-то из участников команды («ну каким дураком надо быть, чтобы такое сделать?!»)
* важность и приоритет бага корректно выставлены;
* к отчеты прикреплены все возможные документы, которые облегчат понимание причины дефекта;
* отсутствуют грамматические и семантически ошибки;
* использован правильных технический язык;
* краткое описание бага должно следовать правилу WWW;
* в ответе д описан только 1 дефект;
* оформление соответствует соглашению о названиях, принятом на проекте;

**Инструмент для отчетов об ошибках** - система создания, хранения и работы с отчетами об ошибках.

## Набор функций:

* создание отчетов о дефектах, управление их ЖЦ с учетом контроля версий, прав доступа и разрешенных переходов из состояния в состояние;
* сбор, анализ и предоставление статистики;
* рассылка. Уведомлений, напоминаний и иных артефактов соответствующим сотрудникам;
* организация взаимосвязей между отчетами о дефектах, тест-кейсами, требованиями и анализ таких связей с возможностью формирования рекомендаций;
* подготовка информации для включения в отчет о результатах тестирования
* интеграция с системами управления проектами.

MANTIS, BUGZILLA, REDMINE, YOUTRACK, PIVOTAL TRACKER;