**Надежность ПО**

Оксана Генадьевна Скиндер (лектор)  
17 лекций  
посещение необязательно  
перед экзом провер конспект  
за лабы 8+ => автомат

# ЛК 1. Тестирование ПО

\* поиск ошибок  
\* проверка соответствия ПО требованиям и здрав. смыслу  
\* оценка работоспособности ПО  
\* способ контролировать качество ПО

**Причины ошибок в ПО:**1. чел. фактор  
2. проблема в описании требований к ПО  
3. недостаток времени  
4. недостаточно продуманная архитектура приложения  
5. недост. знание бизнеса  
6. нехватка профес. навыков и опыта  
7. изменения «в последнюю минуту»

**История тестирования ПО:**

9 сент 1945 оформлен 1й баг  
Первые программы разраб. в рамках программ науч. исследований или программ для нужд министерств обороны.

Тестир. таких продуктов проводилось строго формализовано с записью всех тестовых процедур, тестовых д-х, полученных рез-тов.

Тест. выделялось в отд. процесс, кот. начинался после завершения кодирования, но при этом, как правило, выполнялось тем же персоналом.

В 1960-х много внимания уделялось «исчерпывающему» тестированию, кот. дб проведено с исп. всех путей в коде или всех возможных входных д-х.

Однако это невозможно:  
#1 кол-во возмож. вход. д-х очень велико  
#2 сущ. мн-во путей  
#3 сложно найти проблемы в архит. и спецификациях

Итог: «исчерпыв.» тест. было отклонено и признано теоретически невозможным.

В нач. 1970-х тест. ПО обозначалось как «процесс, направл. на демонстрацию корректности продукта» или как «деят-сть по подтверждению правильности работы ПО»

Впоследствии этот метод тест. был признан неэффективным (лучевая терапия)

Во вт.пол. 1970-х тест. – выполнение программы с намерением найти ошибки, а не доказать, что она работает.  
*Успешный тест – тест, кот. обнаруживает ранее неизвестные проблемы*

В 1980-е тест. расширилось таким понятием, как предупреждение деффектов.  
Стали высказ. мысли, что необх. методология тест-я, в частности, что тест. д. включать проверки на всем протяжении цикла разработки, и это дб управляемый процесс. (м. влиять)

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

В нач. 1990-х в «тест.» включ. планирование, проектирование, создание, поддержку и вып. тестов и тестовых окружений и это означало переход от тестир. к обесп. кач-ва.

Начин. появляться различ. прогр. инструменты для поддержки процесса тестир.: более продвинутые среды для автоматизации с возм-стью создания скриптов и генерации отчетов, системы упр-ния тестами, ПО для проверки нагрузочного тестирования.

|  |  |
| --- | --- |
| Тестирование | Обеспеч. кач-ва |
| готовый продукт есть баги, и мы пытаемся их исправить (тестироващик) | д-вия, чтобы не допустить появление багов (инженер) |

**Активности для обесп. кач-ва:**\* тестир. требований  
\* unit тесты  
\* код-ревью  
\* ранняя интеграция (коммиты 3-4 раз/день)

**ISO** (International Organization for Standartization)  
ISO/IEC 25010 : 2011 ↓

**Качество системы:**Это степень удовлетворенности с-мой заявленных и подразумеваемых потребностей различ. заинтересованных сторон, кот. позволяют т.о. оценить достоинства.

Эти заявленные и подразумеваемые тебования представлены в международ. Станадртах серии SQuaRE посредством моделей кач-ва, кот. представ. кач-во продукта в виде разбивки на классы характеристик, кот. в отд. случаях далее раздел. на подхарактеристики.

**Модели качества:**  
\* модель кач-ва при использ.  
\* модель кач-ва продукта  
\* модель кач-ва д-х ISO/IEC 25012 : 2008

**♥ Модель качества при использовании:**

* Эффективность (результативность)  
  это точность и полнота, с кот. юзеры достигают поставленных целей ($ списались, покупка не оплачена)
* Производительность  
  это связь точности и полноты достижения юзером цели с израсходованными ресурсами (тормозит телефон)
* Удовлетворенность  
  \* полезность (много лишнего: вайбер)  
  \* доверие (МТБанк в чужой кабинет)  
  \* удовольствие  
  \* комфорт (физический: маленькая button)
* Свобода от риска (смягч. отриц. посл-вий)  
  \* экономического  
  \* здоровья/безопасности (мед.)  
  \* экологического
* Покрытие контекста (полнота, гибкость)  
  *Полнота контекста* – степень, в кот. ПО применимо при исп. на маленьком экране с низкой сетевой пропускн. спос-стью неквалифиц. юзерами или в отказоустойч. режиме

*Гибкость контекста* – степень, в кот. продукт мб использован с эффективностью, рез-тивностью и свободой от риска в соотв. с требованиями в условиях, выходящих за рамки изначально определенных

**♥ Модель качества продукта:**

* Функцион. пригодность  
  степень, в кот. продукт обесп. вып-ние ф-ций в соотв. с заявленными требованиями  
  \* ф. полнота – степень покрытия ф-ций всех опр. задач и целей юзера (после $ на стр. одобрения)  
  \* ф. корректность (cancel – отмена)  
  \* ф. целесообразность – степень ф. упрощения вып-ния задач юзера (в соц. сети – кто брат? 2-й email?)
* Уровень производительности  
  степень пр-сти ПО относ. суммы использованных ресурсов  
  \* временные характеристики (30 сек page open)  
  \* исп-ние ресурсов (все голубое: глаза, память, трафик)  
  \* потенц. возм-сти – степень соотв-вия требованиям предельных значений параметров продукта (1000 гостей)
* Совместимость  
  \* сосущ-вание – степень, в кот. ПО м. сосущ. с другим на одном девайсе/сервере  
  \* интероперабельность – степень, в кот. 2+ с-мы могут обмен. инфой (email берет погоду с др. сайта)
* Удобство используемости  
  \* опр. пригодности (как быстро юзер поним, что прил. поможет достиг. цель)  
  \* изучаемость (прил. учебные: наск. понятно)  
  \* управляемость (юзер не м. удалить страницу)  
  \* защищенность от ошибок юзера (недопуст символы в пароле)  
  \* эстетика польз. интерфейса  
  \* доступность – возм-сть исп-я системы для достиж. опр. цели в указ. контексте исп-я широким кругом людей с самыми разными возм-ми
* Надежность  
  \* завершенность (зрелость) (законч память, х интернета – все варианты надежны)  
  \* готовность (раз/месяц 1000 чел на сайте)  
  \* отказоустойчивость (с-ма сама восст-ся)  
  \* восст-мость (спос-сть восст. д-е)

Готовность – доля общего времени, в теч. кот. прил. в работоспос. сост.

* Защищенность  
  \* конфиденциальность  
  \* неподдельность  
  \* целостность  
  \* подлинность  
  \* отслеживание

Конфед. – обесп. и ограничение доступа к опр. ресурсам приложения в соотв. с ролью юзера

Целостн. – степень предотвращ. с-мой несанкц. доступа или модификации д-х (ali.com/admin -> 403)

Неподдельн. – степень, в кот. мб доказан факт события/д-я т.о., что этот факт не мб опровергнут (id)

Отслежив. – степень, до кот. д-вия обхекта мб отслежены однозначно

Подлинность – степень достоверности того, что объем/ресурс = требуемому объекту/ресурсу (на этапе аутентификации – отпечаток пальца)

* Сопровождаемость  
  \* модульность (прил. на модули)  
  \* возм. многократ. исп-я (меню не для к. страницы)  
  \* тестируемость  
  \* анализируемость (наск. быстро ошибка)  
  \* модифицируемость (для исправл ошибок)
* Переносимость  
  \* адаптируемость (шторы сверху: новый IOS)  
  \* устанавливаемость (обновление -> наложение сверху)  
  \* взаимо-заменяемость

Взаимо-замен. – спос-сть продукта заменить другой конкр. прогр. продукт для достиж. тех же целей в тех же условиях

**♥ Модель качества д-х ISO/IEC 25012:2008**

**Терминология:  
DEV Environment** – среда, где разработчики отслеживают, проверяют свои изменения  
**QA Environment** – 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 предъявл. требованиям, кот. осущ. 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** – структура, согласно к-й построено создание нового ПО  
**Жизн. цикл ПО** – период времени, кот. начинается с момента принятия решения о необх. создания прогр. продукта и заканч. в момент его полного изъятия из эксплуатации.

**Составляющие процесса:**  
\* требования \* имплементация  
\* архитектура \* тестирование  
\* дизайн \* релиз

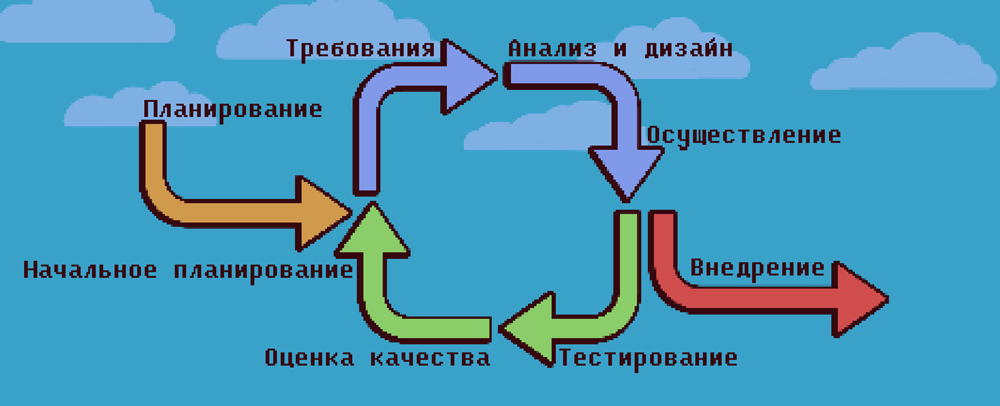
**Модели разработки ПО** – структура, систематизирующая различные виды проектной деят-сти, их взаимод-е и посл-сть в процессе разработки ПО:  
\* каскадная  
\* V-модель  
\* итеративная  
\* инкрементальная  
\* спиральная

**♥ Каскадная модель:**  
В соврем. модели на каждом этапе + тестирование  
подходит: медиц., военное ПО; крупные проекты; распределенная команда  
не подходит: часто меняются требования, небольшие проекты, небольшая команда

1. Тестирование 🡪 2. Дизайн 🡪 3. Кодирование 🡪 4. Тестирование 🡪 5. Поддержка

**♥ V-модель:**  


**♥ Итеративная модель:**  
подходит: небольшие проекты, небольшие к-ды, к-да в одной локации  
не подходит: распредел. к-да, медиц., военное ПО  
! Всегда есть возм-сть в последнюю минуту сделать изменения !

Требования – Дизайн – Имплементация – Тестирование – Релиз   


**AGILE:** философия разработки ПО. Agile-манифесты:  
\* люди и взаимодействие важнее процессов и инструментов  
\* работающий продукт важнее исчерпывающей документации  
\* сотрудничество с заказчиком важнее согласования условий контрактов  
\* готовность к изменениям важнее следования первонач. плану

# ЛК 3. Процесс тестирования

**Процессы тестирования:**1. Планирование и управление  
2. Анализ и проектирование  
3. Внедрение и реализация  
4. Оценка критериев выхода и создание отчета  
5. Д-вия по завершению тестов

**1) Планирование тест.** – д-вия, направленные на: определение целей и способов их достижения  
1. анализ требований (что д. делать прил и мы)  
2. опр. целей тестир (найти баги/шаги/проверить фичу)  
3. опр. общего подхода к тестир (уровни, виды тестир, критерии входа)  
4. интегрирование с разработкой ПО  
5. реш., какие роли нужны в тестир (автоматизаторы, мануалы)  
6. сост. график  
7. опр. шаблоны для тестовой документации  
8. выбор метрик для мониторинга и контроля тестир

**Управление тестир.** – постоянное сопоставление тек. положения дел с планом и отчетность о состоянии дел, включая отклонения от плана.

**Критерии входа:**опр., когда нужно начать тестирование  
\* готовность и доступность тестового окружения (clear environment)  
\* готовность ср-в тестирования в окружении  
\* доступность тестир. кода  
\* доступность тестовых д-х

**2)** **Анализ и проектирование**  
деят-сть, во время кот. цели тестир. материализуются в тестовые условия и тест. сценарии

\* Рецензирование базиса тестир.:  
 (не)функц. требования, архитектура, дизайн, техн. требования к интерфейсу  
\* Оценка тестируемости базиса и объектов тест  
\* Идентиф. и расстановка приоритетов тест (1.проверить позитив сценарии и негативные!)  
\* Выявление необх. д-х для поддержки тестовых условий и тест. сценариев  
\* Проектир. и установка тест. окружения и выявления необх. инфрастр-ры и инструментов  
\* Создание 2-направл. трассируемости между тест. базисом и тест. сценарием  
 (к. треб <--> одна проверка)

**3)** **Внедрение и реализация**  
это деят-сть, где процедуры тест. или автоматизированные сценарии задаются посл-стью тест. сценариев, + собир. любая инфа, необх. для вып. тестов  
разворач. окр. среда и запуск. тесты

\* заверш. реализация и расстан. приоритетов тест. сценариев  
\* написание автоматиз. тесты  
\* проверка правильности настройки тест. окружения  
\* проверка и обновл. 2-напр. трассируемости между тест. базисом и тест. сценарием  
\* вып. процедур тестир. либо вручную, либо исп-я инструменты вып. тестов, согласно плану  
\* регистрация результатов  
\* сравнение фактического и ожидаемого рез-тов  
\* оформл. отчетов об ошибках и занесение их в баг-трекинговую сис-му  
\* повторное тестир областей, где были исправлены ошибки и областей, где могут появ. новые после исправления старых

**4)** **Оценка критериев выхода и созд. отчета**  
это деят-сть, где вып. тестов оценивается согласно опр. целям  
Она дб выполнена для каждого уровня тестир.

\* сверка протокола тест. в сравн. с критериями выхода, опр. в плане тестир  
\* анализ необх-сти исп. доп. тестов или изменения критериев выхода  
\* написание итог. отчета о тестир. для заинтерес. лиц

**Критерии выхода**опр., когда нужно прекращать тестирование  
\* степень покрытия кода, функ-сти или рисков тестами  
\* оценка плотности дефектов (измерение надежности)  
\* стоимость  
\* остаточные риски (неисправл. дефекты или недостаток тест. покрытия области)  
\* план, осн. на времени выхода ПО на рынок

**5)** **Действия по завершению тестов**  
это сбор д-х о заверш. испытаниях, для объед. опыта, тестового обесп., фактов и цифр. После релиза  
\* проверка, что запланир. рез-ты достигнуты  
\* заверш. и архивирование тест. обесп., тест. окр-я и инфрастр-ры тест. для послед. исп-я  
\* передача тест. обесп. организации сопровождения  
\* анализ получ. уроков для повыш. зрелости процессов тест.

|  |  |
| --- | --- |
| **Процесс разработки** | **Процесс тестир.** |
| Анализ требований Разраб. архитектуры Проектирование, дизайн Имплементация Отладка  Инсталляция приложения на Production | Анализ требований Планирование тестир. Анализ и проект. тестов Реализация и вып. тестов Реал. и вып. тестов Оценка критериев выхода и отчет Д-я по заверш. тестир |

|  |  |
| --- | --- |
| **Процесс тестир.** | **Процесс обесп. кач-ва** |
| Планир и управление Анализ и проектир Внедр и реализация Оц. критериев выхода и созд отчета Д-вия по заверш тестов  (это все alpha-тестир) | Тест. требований Релиз. unit-тестов Релиз. интеграц. тестов Code review Интеграция фичи в с-му Alpha-тестир Отладка Beta-тестир |

**Уровни тестирование ПО:**  
\* компонентное тест логин, форма поиска, фильтр  
\* интеграционное тест новая фотка – у друзей тоже  
\* системное тест все тесты, вся с-ма целиком  
\* приемочное тест зафрендить вк, письма в mail.ru

В процессе приемоч. тестир. мы вып. валидацию и провер, насколько прилож. соотв. бизнес-цели

*Компонентное:*  
Каталог товаров, Поиск товаров, Оплата, Профайл покупателя

*Интеграционное:*  
Каталог + Поиск, Оплата + Профайл

*Системное:*  
Каталог + Поиск + Оплата + Профайл

# ЛК 4. Типология тестир

**2 метода тестирования:**

* метод белого ящика – тестируется только код (бэкэнд прилож)
* метод черного ящика – тестирование через интерфейс программ (нет исх. кода)

**Типы тестирования** – группы активности тестирования, кот. направлены на проверку работоспособности с-мы, где за основу принимаются различ. цели и причины тестир

**Типы тестирования**

**1) функциональное тестир**  
такое тестир, кот. обрабатывается на основе функций и возм-стей с-мы, а также их взаимод. с другими с-ми.  
Проводится на разных уровнях;  
Проводится методом черного ящика;

**Примеры:**  
- позитивное тестир  
- негативное тестир  
- тестир CRUD (create, read, update, delete)

**2) нефункцион. тестир**  
проводится для оценки хар-к систем и программ  
Проверяется не корректность работы функций прилож, а соотв. хар-ки

**Примеры:**  
- тестирование внеш вида приложения (методом черного ящика)  
- нагрузочное тестир (методом черного и белого ящика)  
- тестир безопасности (методом черного ящика)  
- тестир совместимости (методом черного ящика)

**3) структурное тестир**  
анализ и тестир кода продукта, его архитектуры  
Проводится методом черного ящика

**Примеры:**  
- unit-тесты  
- интеграционные автоматизированные тесты  
- тестир веб-сервисов

**4) тестир изменений**  
любое повторное тестир уже протестир программ после внесения в них изменений, чтобы обнаружить дефекты, внесенные или пропущенные в рез. этих действий  
Чаще проводится методом черного ящика;

**Примеры:**  
- регрессионное тестир  
- тестир, основанное на рисках

**Виды функционального тестирования**

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

|  |  |
| --- | --- |
| **Позитивное** | **Негативное** |
| тестир, при кот. использ. только валидные д-е и выполн только валидные д-вия | тестир с использ невалидных д-х и д-вий, направл на получение ошибок и предупреждений |
| **Исследовательское** | **Интуитивное** |
| подход к тестир, кот. подразумевает под собой одновременное изучение приложения, проектирование тестовых сценариев и их немедленное выполнение | неподготовленное и недокументированное тестир, «игра» с приложением по свободному сценарию без исследования тестовой документации |
| **Отличие** исслед от интуитивного: \* у исслед 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 никогда нет цели | |
| **End-to-end тестир** | **Role-based тестир** |
| подход, при кот. проводится сквозное тестир с-мы по сценариям исп-ния с-мы реальными юзерами | тестир, направл на проверку ограничения прав доступа к конкр. функционалу на основе ролей и привилегий юзера |
| Осн. **задача тестировщика** – продумать как можно больше разнообразных сценариев исп-ния приложений Существенная **разница** между этими 2 видами: первое имеет одного юзера и затрагивает как можно больше компонентов прилож, а во втором – как можно больше ролей | |
| **Инсталляционное** | **CRUD** |
| тестир, при кот. проверяется корректность инсталляции и настройки прилож, а также обновления и удаления ПО  **Примеры:** \* устан с параметрами по умолч \* устан с измененными параметрами \* устан в silent mode \* устан на разл. поддерж-мые платформы \* устан обновления на работающее ПО \* удаление ПО | тестир корректности работы 4 функциональных возможностей каждой создаваемой записи в прилож: - создание - просмотр - редактирование - удаления |
| Создание и редактирование всегда д. происходить по тем же правилам. **2 вида удаления:** \* soft – скрывается, но ост. в БД \* hard – запись полностью удал из БД |
| **По приоритету** | **Smoke test** |
| виды тестир, направл на выявление кач-ва функционала опред. важности: \* дымовое тестир (самый важный) \* тестир критического пути (внизу) \* расширенное тестир (extended) | тестир, направл на определение и проверку самой важной функциональности. Неработаспособность делает бессмысленной идею исп-ния фичи или прилож. Баги, найденные в smoke-test - **блокирующие** |
| **Critical path test** | **Extended test** |
| тестир, направл на исслед функциональности, исп-мой типичными юзерами в типичной повседневной деят-сти | тестир, направл на исследование всей заявленной в требованиях функциональности – даже той, которая имеет низкий приоритет |

# ЛК 5. Позитивное и негативное

**Метод** тестир отвеч. на вопрос *«Каким способом мы проводим тестир?»*  
\* способ белого ящика – инспектируем ход  
\* способ черного ящика – взаимод с с-мой через интерфейс

**Уровень** отвеч. на вопрос *«Как глубоко мы тестируем с-му?»*  
\* на уровне маленького компонента  
\* на уровне интеграции  
\* на уровне всей с-мы

**Виды** отвеч. на вопрос *«С какой целью мы тестир? Что хотим найти?»*

**Модели поведения польз-ля**

**Пользователь-интуит**  
польз. не читал инструкций или не способен их прочитать. Находится несоответствие интерфейса/поведения проги сущ-щим стереотипам  
\* проводим негативное + позитивное тестир

**«Хороший» пользователь**  
добросовестный польз. д-вует в строгом соответствии с инструкциями. Поиск ошибок как в логике работы проги, так и в документации на прогу  
\* больший приоритет позитивному тестир.

**«Плохой» пользователь**  
недобросовестный польз., стремится использ. прогу непредусмотренным способом  
\* в 1 очередь негативное тест.

**Позитивное тест.** – тестир, при к-ром проверяются корректные варианты исп-ния сис-мы + реакция с-мы на валидные д-е. *Осн*. *цель* – проверка того, что при помощи с-мы можно делать то, для чего она создавалась.

**Позитивное тестирование: примеры**

\* протестир, что выбор *високосного года* проверяется корректно и не приводит к ошибкам и неправильным расчетам  
\* протестир *отриц*. числа в числовом поле (если разрешены)  
\* протестир *дробные* числа в числовом поле (отдел. десятичные ‘.‘ или ‘,’ в завис от локализации)  
\* протестир, что система правильно обраб *деление на 0*  
\* протестир *макс*. длину поля что данные не обрезаются  
\* протестир функциональность *Timeout* (веб сессия – разлогин ч/10 мин)  
\* протестир *сортировку* (в таблице) (спецсимволы -> числа -> алфавит; дата: год->мм->дд)  
\* протестир функциональность *кнопок*\* провер, что *Privacy Policy* & FAQ доступны для юзеров (в футере)  
\* протестир, что загруженные *файлы* можно открыть / скачать  
\* провер все *имейлы*, кот. генерирует система (письма дб не пустые)  
\* провер, что *JS* работает правильно во всех поддерживаемых браузерах (~~IE~~, Firefox, Chrome, Safari)  
\* подтвержд. сообщение (*confirmation* *messages*) должны исп. один и тот же CSS стиль, и он д. отличаться от стиля сообщений об ошибке (зеленый цвет)  
\* провер *тултипы* (их знач и текст) (=всплыв подсказки: в разных браузерах + всё отображ)  
\* провер *push-нотификации*, текст в них и корректность работы (жмем на сообщ – откр диалог с челом)  
\* первое знач в *dropdown* списке дб или пустым, или ‘Select’ ( или др. соотв-щее контексту) (выбор пола!)  
\* должны присут опции *Select*/deselect all для чекбоксов (при наличии >10 чекбоксов)  
\* провер *надписи* над полями  
\* провер *сортировку* после добавления, апдейта или удал какой-либо записи

**Негативаное тестир** – тестир, целью к-го явл. проверить, что система корректно обраб. невалидные д-е/д-вия пользователя и в понятной юзеру форме дает обратную связь

**Негативное тестирование: примеры**

\* протест, что все *обязательные* *поля* проверяются системой  
\* протест, что система не выдает валидационную ошибку на *необязательные* *поля*  
\* протест, что юзер не может ввести *больше*/*меньше* символов чем допустимо  
\* проверить, что если происходит какая-то ошибка (*404* или *500*, или др), то система перенаправляет юзера на спец. страницу приложения, где нах. предупреждающее сообщение (application crash or unavailable pages)  
\* протест, что случается, если юзер *удаляет куки* после посещения сайта  
\* проверить расположение *валидационного сообщения*  
\* все сообщ об ошибках д. исп один и тот же *CSS* *стиль* (красный цвет)  
\* ввод д-х с *пробелами* в начале, середине и в конце

**Тестирование формы логина**

|  |  |
| --- | --- |
| Позитивное тестирование | Негативное тестирование |
| \* при откр формы фокус дб в поле Username \* поле Password дб типа “Password” \* ввести сущ-щий в с-ме Username, ввести правильный Password, нажать “Enter” \* ввести Username, Password и нажать Cancel | \* оставить поля пустыми и нажать “Enter” или Log in \* ввести несущ-щие в системе Username, ввести неправильный Password, нажать Log in |

**Тестирование формы регистрации**

|  |  |  |
| --- | --- | --- |
|  | Позитивное тестирование | Негативное тестирование |
| Полн.имя | \* имя с пробелами \* имя с дефизом и апострофом \* имя в верхнем и нижнем регистре \* имя в пределах мин и макс | \* оставить поле пустым \* имя за пределами мин и макс \* спецсимволы вместо имени |
| Email | \* e-mail в корректном формате \* уникальный e-mail \* e-mail в пределах мин и макс | \* оставить поле пустым \* адрес, кот. не соотв. формату \* спецсимволы вместо e-mailа \* e-mail, кот. уже использ. |
| User Name | \* Username в корректном формате \* уникальный Username \* E-mail в пределах мин и макс | \* оставить поле пустым \* Username, кот. не соотв. формату \* Username с пробелами \* Username, кот уже использ \* Username, кот за пределами мин и макс |
| Password | \* Password отобр точками/звездочками \* Password в пределах мин и макс \* Password соотв формату | \* оставить поле пустым \* Password, кот не соотв. формату \* Password, кот за пределами мин и макс |
| Соц.сети | \* нажать на кнопку соц сети, когда сессия в ней открыта \* нажать на кнопку соц сети, когда сессия в ней не открыта \* проверить возм-сть вернуться назад на форму регистрации | \* попытаться зарегаться одновременно используя несколько соц сетей |

**Тестирование даты**

|  |  |  |
| --- | --- | --- |
|  | Позитивное тестирование | Негативное тестирование |
| Календарь | \* выбор тек. даты \* выбрать даты в будущем \* перелист 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 дату «Назад» раньше, чем «Туда» \* выбрать корректно даты «Туда» и «Назад», потом выбрать дату «Туда» позже чем «Назад» \* ввести дату с клавиатуры |
| Чеклист | \* выбор корректной даты в прошлом \* ввод даты с клавиатуры \* выбор тек. даты | \* оставить поля пустыми \* выбрать 31 число и месяц апрель/июнь/сентябрь/ноябрь \* выбрать 29 февраля 1989 года \* выбрать дату в будущем \* набрать некорректную дату с клавиатуры |

**Тестирование № телефона**

|  |  |
| --- | --- |
| Позитивное тестирование | Негативное тестирование |
| \* ввод корректного № телефона с пробелами \* ввод корректного № телефона с дефизами \* ввод корректного № телефона со ( ) \* ввод номера разных операторов \* выбор другой страны (измен код страны) \* смена IP адреса (измен страна по умолч) | \* оставить поля пустыми \* ввести спецсимволы или буквы \* ввести телефон за пределами мин и макс \* написать телефона, потом сменить страну \* набрать номер, сост. из 0 |

**Тестирование строки поиска**

|  |  |
| --- | --- |
| Позитивное тестирование | Негативное тестирование |
| \* провер, что placeholder исчез, когда фокус в поле \* поиск по нажатию на кнопку Search \* поиск по нажатию “Enter” \* поиск соотв-щих страниц \* поиск с применением операторов \* вставка текста поиска из буфера обмена \* поиск по неск. словам \* Autocomplete | \* оставить поле пустым и нажать Search \* поиск по спецсимволам \* текст поиска за пределами мин и макс \* поиск страниц, к-рых не сущ. |

# ЛК 6. Исследовательское

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

**Цели исслед. тестирования:**  
\* проверить с-му с т.зр. конечного юзера  
\* найти баги, кот. не были найдены путем формального процесса тестирования  
\* изучить работу приложения, опираясь не только на документацию, но и на поведение с-мы после манипуляций юзера

**Процесс исслед. тестирования:**

1. Подготовка
2. уточнение feature under test
3. уточнение цели тестир (mission)
4. уточнение окружения (платформы/браузеры/у-ва)
5. уточнение продолжительности сессии
6. уточнение, какая документация необх. до/после тестир
7. Тестирование / оформление документации
8. Дебрифинг (de-briefing)

**Дебрифинг** – краткий рассказ заинтересованным лицам о том, как проходило тестирование, какие проверки выполнялись, с какими проблемами столкнулись, на какие области в будущем надо обратить внимание.

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

**Продолжительности сессий:**  
\* short (60 мин) //чтобы вспомнить что обсудить  
\* normal (90 мин)  
\* long (120 мин)

**Документация в ИТ:**  
\* checklist (общий, краткий список с проверками)  
\* test charter (подобный док с деталями тест-сессии)  
\* записи шагов тестировщика в приложении (с пом. ПО типа Jira Capture, Session tester, qTrace…)  
\* bug reports (отчеты об ошибках)  
\* логи приложения (если уместно)

**TEST CHARTER: Структура док-та** (жирные поля обязательны!)

* Test Charter **ID** (уник идент)
* **Charter** name (краткое название: Rained-out tour, Login Form Critical Path Test)
* **Mission** (цель сессии: 3-5 предложений)
* **Areas** (список компонентов/модулей, кот. будут затронуты тестированием)
* **Test Environment & URL** (ссылка на application under test, ОС/браузеры/у-ва, где будет проходить тестир)
* **Start date & time**
* **Tester** name
* **Duration** (short, normal, long и кол-во минут)
* **Testing notes** (список проверок, что планируется проверять)
* Data files (если требуется)
* **Bugs** (название (title) багов и их ID)
* Issues (описание трудностей при тестир, если были)
* Session Setup Time (время потрач на подготовку, напр. установку new билда на моб. у-во)
* Test Design & Test Execution (% потраченный на продумывание проверок и на само тестирование)

**Тур** – это сессия ИТ, кот. имеет определенную цель  
**Районы** – это группы туров, раздел. по определенной тематике

|  |  |  |
| --- | --- | --- |
| Район: |  | Путешествия: |
| *Бизнес район* | все туры, связ. с бизнесом приложения | \* по путеводителю (по мануалу)  - блоггера (искать что пишут юзеры)  - недовольного всезнайки (по негат отзывам)  \* денежное (тест областей, кот приносят $)  - скептика (тест непопулярных способов оплаты, дост-я цели)  \* по ориентирам (тест заранее оговоренных областей прилож)  \* интеллектуала (цель – задавать сложные вопросы прил-ю)  - заносчивого амера (созд. записи в прилож со специф. д-ми)  \* почты FedEx (тест как д-е созд в с-ме, где они хран, как они трансформ)  \* “после работы” (провер различ. задачи, вып. на сервере в нераб. t прилож)  \* мусорщика (проверка самых низкоприор фич) |
| *Историч.* | связан с пред. версиями или областями, кот. давно не тестировались | \* с “плохим” соседом (не раскрыв меню – не м. тест др. фичу)  \* по музеям (тест областей, в кот. месяц+ не заходили)  \* по уже посещенным местам (тест областей, кот. работали в пред. версиях) |
| *Туристич.* | туры, не связ. с осн. целями прилож (бонстики) | \* суфлера (тест альтерн. способов вып. д-вий – много кнопок)  \* по переулкам (околоважные вещи, не принос $: избранное)  \* “по ночам” (ост прил запущенным на длит t: на ночь) |
| *Развлекат.* | фичи, имеющие более развлек. цель и имеют второстеп. хар-р (лайки) | \* коллекционера (созд всех записей, возможных в с-ме)  \* одинокого бизнесмена (тест самый длинный путь вып. д-й)  \* супермодели (тест интерфейса приложения)  \* test 1 get 1 free (тест изменений одной и той же записи из разных окон браузера/девайсов)  \* по шотландским пабам (тест модулей, про к-е мало кто знает) |
| *Р-н отеля* | все туры, связ. с минимальным кол-вом д-вий, юзер не хочет напрягаться | \* испорченный из-за дождя (тест отмены всех д-вий возможных в системе)  \* ленивца (провер, что записи созд с мин кол-вом символов) |
| *«Плохой» район* | любые вредоносные д-вия юзера | \* саботажника (негативное тест: не валидные д-е)  \* антисоциальное (хакер. атаки)  \* чела, склонного к повтор. д-виям (100 нажатий) |

# ЛК 7. Тестир требований

**Требование** – описание того, какие ф-ции и с соблюдением каких условий должно выполнять приложение в процессе решения полезных для юзера задач.

**Важность требований:**  
\* позв. понять, что и с соблюдением каких условий система должна делать  
\* предоставляет возможность оценить масштаб изменений и управлять изменениями  
\* явл. основой для формирования плана проекта (в том числе плана тестирования)  
\* помогают предотвр или разрешать конфликтные ситуации  
\* упрощ. расстановку приоритетов в наборе задач. Позволяют объективно оценить степень прогресса в разработке проекта

Чем позднее исправим ошибку – тем дороже она будет стоить.

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

* в виде спецификации (схемы, как с-ма работает, как выглядит?)
* техническое задание (опис. техн. реализацию: какие классы, методы)
* варианты использования (как худ. произведение: Бен – отец двоих детей, он жмет на кнопку…)
* user story (как админ, я должен иметь возможность…)
* mockups/wireframes (схема: какая посл-сть, расположение эл-тов)
* дизайн (уже видим явное расположение всего, видим эталон)

**Уровни и виды требований:**  
\* бизнес-треб  
\* пользовательские треб  
\* бизнес-правила  
\* функциональные треб  
\* нефункциональные треб  
\* треб к интерфейсам  
\* треб к данным

**1. Бизнес-требования** – выражают цель, ради кот. разрабатывается продукт. Результат выявления требований на этой уровне – это общее видение – документ, кот. , как правило, представлен простым текстом и таблицами. Нет детализации поведения системы и иных технических хар-к, но мб определены приоритеты решаемых бизнес-задач, риски и т.д.  
Исп. для проведения валидации и приемочного тестирования. Примеры:

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

**2. Пользовательские требования** – описывает задачи, кот. юзер может выполнять с пом. разрабатываемой системы (реакцию системы на д-вия юзера, операции работы юзера). Т.к. появляется описание поведения системы, требования этого уровня мб использованы для оценки объема работ, стоимости проекта, времени разработки и т.д.

* Варианты использования
* Пользовательских историй
* Пользовательских сценариев

Используются на системном уровне.

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

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

* В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе
* Система должна авто- выполнять резервное копирование данных ежедневно в указанный момент времени
* Эл. адрес юзера, вводимый при регистрации, должен быт проверен на соответствие требованиям RFC822

Используются часто на компонентном и интеграционном уровнях тестирования

**5. Нефункциональные требования** – описывают свойства системы (удобство использования, безопасность, надежность, расширяемость и т.д.), которыми она должна обладать при реализации своего поведения. Здесь проводится более техническое и более детальное описание атрибутов качества. В контексте проектирования нефункц. требования в основном влияют на архитектуру системы.  
Используются на компонентном, интеграционном и системном уровнях. Используются для видов нефункционального тестирования.

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

**6. Требования к интерфейсам** – описывают особенности взаимодействия разрабатываемой системы с другими системами и ОС.

* Обмен д-ми между клиентской и серверной частями приложения при осущ. фоновых AJAX-запросов дб реализован в формате JSON
* Протоколирование событий должно вестись в журнале событий ОС
* Соединение с почтовым сервером должно выполняться согласно RFC3207 (“SMTP over TLS”)

**7. Требования к данным** – описывают структуру данных (и сами данные), являющиеся неотъемлемой частью разрабатываемой с-мы. Часто сюда относят описание БД и особенностей ее использования.

* Все данные системы должны храниться в БД  
  польз. документы – под управлением СУБД MongoDB  
  не польз. документы – под управлением СУБД MySQL
* Информация о кассовых транзакциях за текущий месяц должна храниться в операционной таблице, а по завершению месяца переноситься в архивную

**Баги в требованиях:**

* Противоречия (в разных частях спецификации разная инфа)
* Неоднозначности (user-friendly: хлебные крошки, комменты)
* Неполные требования (с-ма не д. давать юзеру ввести неправильный логин/пароль)
* Отсутствие единого стиля в работе ПО (сообщ об ошибке то сверху, то снизу)
* Невозможность реализовать требования (скроллинг в браузере розовый)
* Отсутствие трассируемости (в неск. документов разная инфа)

**Способ уточнения требований:**

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

Баги в требованиях оформляются в виде вопросов и по возможности уточняются

**Оформление вопросов:**

* Вопросы пишите короткими и простыми (да/нет)
* Вопрос не должен содержать «или»
* Формулируйте вопрос так, чтобы ответ на него был максимально коротким
* Обдумайте ответ, который можно получить
* Предлагая улучшения, подкрепляйте их фактами с весомыми доводами (поменять домен с ru,by на com)
* Просьбы подкрепляйте вежливыми оборотами

**Примеры:**  
**Требование**: приложение должно быстро запускаться --неоднозначность

**«Плохие» вопросы**:  
*Насколько быстро?  
А если не получится?*

**«Хорошие» вопросы:** *Какое макс допустимое врем запуска приложения?  
Допуск ли фоновая загрузка отд. компонентов приложения?  
Что является критерием того, что приложение закончило запуск?*

**Требование:** --неоднозначность + неполные требования  
Опционально должен поддерживаться экспорт доков в формат PDF

**«Плохие» вопросы:**  
*«Любых документов?»*  
(Ответы: «да, любых» или «нет, только открытых» вам все равно не помогут)

*«В PDF какой версии должен производиться экспорт?»*  
(Вопрос хороший, но он не дает понять, что имелось в виду под «опционально»)

*«Зачем?»*  
(«Нужно!» Именно так хочется ответить, если вопрос не раскрыт полностью)

**«Хорошие» вопросы:**  
*«При каких условиях будет доступен экспорт в PDF?»  
«Для каких документов будет поддерживаться экспорт?»  
«Какие действия юзеру необходимо выполнить, чтобы экспортировать документ в PDF?»*

**Требование:**  
Если дата события не указана, она выбирается автоматически

**«Плохие» вопросы:** *«А если указана?»*  
(То она указана. Логично, не так ли?)

*«А если дату невозможно выбрать автоматически?»*  
(без пояснения причин невозможности звучит как издёвка)

*«А если у события нет даты?»*(Автор вопроса, скорее всего, хотел уточнить, обязательно ли это поле. Но из самого требования видно, что обязательно)

**«Хорошие» вопросы:**  
*«Возможно, имеется в виду, что дата генерируется автоматически, а не выбирается?»*Если *«Да»,* то по какому алгоритму генерируется?  
Если *«Нет»,* то из какого набора выбирается дата и как генерируется этот набор?

# ЛК 8. Нефункц виды тестир

**Нефункц. тестир** – тест, кот. проводится для оценки хар-к ПО. Проверяется не корректность работы функций приложения, а сопутствующие хар-ки.

1. тестир производительности
2. безопасности
3. эргономичности *(usability testing)*
4. совместимости *(с браузерами, платформами)*
5. UI тестирование
6. локализации и интернационализации
7. A/B тестирование
8. на отказ и восстановление
9. на соответствие стандартам
10. на прерывания *(работы мобильного ПО)*
11. соединения *(работы мобильного ПО)*

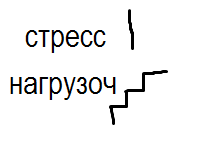
**I) ТЕСТИРОВАНИЕ ПРОИЗВ-СТИ** – комплекс тестов, определяющих поведение приложения при нормальной и предельной нагрузках.

**Подвиды нагрузочного тестирования:**  
1) тестир производительности  
2) стресс-тестирование  
3) нагрузочное  
4) стабильности  
5) на больших объемах данных

**1. Тестир произв-сти** – тестир, кот. определяет, как быстро работает ВС или ее часть под определенной нагрузкой (измеряется на стороне сервера)  
Пример: скорость генерации учета по опр. параметрам из фильтра

**2. Стресс-тестир** – тестир проводится для определения надежности с-мы во время экстремальных или диспропорц. нагрузок и отвеч на вопросы о достаточной произв-сти с-мы в случае, если тек. нагрузка сильно превысит ожидаемый максимум.  
резкое (почти единовременное) повышение нагрузки до макс величин;  
Пример: начало трансляции футбольного матча

**3. Нагрузочное тестир** – тестир, кот проводится для того, чтобы оценить поведение приложения под заданной ожидаемой нагрузкой

Этой нагрузкой мб ожидаемое кол-во одновременно работающих юзеров приложения, совершающих заданное число транзакций за интервал времени

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

В нагрузочном тестир юзеры добавляются постепенно и аналитик произв-сти смотрит, с какого кол-ва юзеров система начинает деградировать;

**Процесс нагрузочного тестирования:**

1. Анализ требований к ПО  
   - ожидаемое кол-во одновременно раб-щих юзеров  
   - ожидаемое время отклика с-мы при макс нагрузках  
   - ожидаемое состояние с-мы при макс нагрузках  
   - ожидаемое поведение с-мы при макс нагрузках
2. Определение основных сценариев исп-ния приложения  
   по ним будут бегать виртуальные юзеры
3. Запись и отладка скриптов (сценариев)
4. Запуск скриптов с необх. кол-вом юзеров и снятие показателей
5. Анализ причин возникших проблем в приложении,  
   подготовка возможных решений
6. Внедрение решений и повторный запуск тестов  
   (п. 5 и 6 повторяют, пока проблемы с произв-стью не решатся)

**4. Тестир стабильности** – проводится с целью убедиться, что приложение выдерживает ожидаемую нагрузку в течение длительного времени (от 2 недель до месяца)

Осущ. наблюдение за потреблением приложением памяти, чтобы выявить потенциальные утечки. Выявляет деградацию произв-сти, выражающуюся в снижении скорости обработки инфы и/или увеличении времени ответа приложения после продолжительной работы по сравнению с началом теста  
Нагрузка дб в течение всего времени максимальной

**5. Тестир на больших объемах д-х**  
тестир произв-сти с-мы на больших объемах д-х  
*(хештег #любовь #навсегда)*

**II) ТЕСТИР БЕЗОПАСНОСТИ** – комплекс тестов, определяющих степень уязвимости ПО к различным атакам

**III) ТЕСТИР ЭРГОНОМИЧНОСТИ** – исследование с целью опред, удобен ли некоторый искусственный объект (веб-страница, польз. интерфейс или у-во) для его предполагаемого применения. Это метод оценки удобства продукта в исп-нии, основанный на привлечении юзеров в качестве тестировщиков, испытателей и суммировании полученных от них выводов.

Юзеры должны входить в целевую аудиторию приложения, иначе все его улучшения эргономичности не считаются значимыми  
\* “кнопка слишком большая” -> на маленькую нельзя нажать старикам  
\* клик на рекламу -> сразу видно магазин

**IV) ТЕСТИР СОВМЕСТИМОСТИ** – тестир, целью кот. является проверка корректной работы приложения в определенном окружении (провер, что всё видимое на своих местах, анимации…)

* аппаратная платформа
* сетевые ус-ва
* периферия (принтеры, CD/DVD-приводы, веб-камеры)
* ОС (Unix, Windows, Mac OS)
* БД (Oracle, MS SQL, MySQL)
* Системное ПО (веб-сервер, файрволл, антивирус)
* Браузеры (IE, Firefox, Opera, Chrome, Safari)

**V) UI ТЕСТИРОВАНИЕ** – тестир соответствия внешнего вида продукта заявленным дизайнам и требованиям.  
Мы проверяем корректность исп-мого шрифта (тип, размер), корректное применение стилей (маржинг, паддинг) – всё, на что не влияет браузер (считать пиксели)

**VI) ТЕСТИРОВАНИЕ ГЛОБАЛИЗАЦИИ**

* **Локализация ПО**  
  процесс адаптации ПО к культуре какой-либо страны. Как частность – перевод UI, документации и сопутствующих файлов ПО с одного языка на другой
* **Интернационализация ПО**  
  технологические приемы разработки, упрощающие адаптацию продукта к языковым и культурным особенностям региона, отличного от того, в кот. разрабатывался продукт
* **Отличие локализации от интернационализации:**  
  *интернац* – на нач. этапах разработки, *локализ* – для каждого целевого языка

**Локализ:** главный вопрос : “Все ли элементы страницы переведены?”  
 ошибка: название страницы не переведено, плейсхолдер, текст на картинках  
 свинина для евреев в онлайн-магазине  
 арабские эмираты: текст справа-налево, картинки детей и женщин

**Интерн:** специфический формат хранения д-х, универсальный для поддерживаемых регионов  
 дата, время (am/pm, 24), разделители в числах, суммах, валютах  
 в мобильных – счит. IP или системный язык в настройках  
 в браузерах – счит. браузерную локаль из системы

**VII) A/B ТЕСТИРОВАНИЕ**

метод маркентингового исследования, суть кот заключается в том, что контрольная группа элементов сравнивается с набором тестовых групп, в которых один или несколько показателей были изменены, чтобы выяснить, какие из изменений улучшают целевой показатель

**VIII) НА ОТКАЗ И ВОССТАНОВЛЕНИЕ**

тестир, кот. проверяет продукт с т.зр. способности противостоять и успешно восстанавливаться после возможных сбоев, возникших в связи с ошибками ПО, отказами оборудования или проблемами связи. Цель – проверка систем восстановления, кот. в случае сбоев, обеспечат сохранность и целостность д-х тестируемого продукта

**IX) ТЕСТИРОВАНИЕ ПО СТАНДАРТАМ**

процесс тестир для определения соответствия компонента или системы стандартам, нормам и правилам  
\* ISO/IEC 25010:2011  
\* Apple guidelines  
\* Google play guidelines

**X) ТЕСТИРОВАНИЕ НА ПРЕРЫВАНИЕ**

тестир мобильного ПО на корректность обработки различных прерываний в работе приложения  
\* входящие звонки  
\* СМС сообщения  
\* системные сообщ (напр., низкий уровень заряда батареи)  
\* screen-lock  
\* уход в спящий режим

**XI) ТЕСТИРОВАНИЕ СОЕДИНЕНИЯ**

тестир мобильного ПО на корректность работы на различных типах соединения  
(пропала связь (печатали сообщ), потом появилась – сообщ авто- сразу отправляется)  
\* 2G, 3G, 4G  
\* Wi-fi  
\* EDGE  
\* No connection

# ЛК 9. Автоматизация

**Автоматизированное тестир** – часть процесса тестир на этапе контроля кач-ва в процессе разработки ПО. Оно использует программные средства для выполнения тестов и проверки рез-тов выполнения, что помогает сократить время тестир и упростить его процесс

Относится к **функциональному** виду тестир и проверяет корректность работы отдельных методов/компонентов/всего приложения

**Уровни авто-тестир:**  
\* уровень модульного тестир (unit tests)  
\* уровень функционального тестир (non-UI-tests)  
\* уровень тестир через польз. интерфейс (UI tests)

**1. Модульное тестирование**

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

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

**Пример:**  
class CalculatorTests {

public void sum\_2plus5\_7returned() {

var calc = new calculator(); //arrange

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

assert.areequal(7, res); //assert

}}

**Правила UNIT-тестов**

Тесты должны быть:  
\* достоверными  
\* не зависеть от окружения, на кот. они выполняются  
\* легко поддерживаться  
\* легко читаться и быть простыми для понимания  
 (даже новый разработчик д. понять что именно тестируется)  
\* соблюдать единую конвенцию именования  
\* запускаться регулярно в автоматическом режиме  
\* один тест должен проверять только одну сущность  
\* тесты должны загружаться в с-ме контроле версий  
\* названия дб «говорящими»

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

class Calculator tests {

private 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 тестир**

**NON-UI тестир** – процесс в программировании, позволяющий проверить работоспос-сть приложения, используя программный интерфейс приложения.

**Пример:**  
package CountriedRestTests;  
…..  
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-тестир** – процесс в программировании, позвол. проверить работоспос-сть 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 .  
\* созд карт объекта реализация фреймворка  
\* созд набора ключ.слов обеспеч. ключ слова

|  |  |  |
| --- | --- | --- |
| Объект | Действие | Данные |
| Login page | 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  
 o выбор системы автозапуска тестов  
 o 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. Тестир 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) – 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. Тест документация

**Тестовая документация** – док., *создаваемая тестировщиками*, кот. помогает в выполнении различного рода активности в рамках тестирования ПО

**Проектная документация:** требования - (не)функц., польз., бизнес-треб.; проектный график; проектный план; различ. дедлайны и правила проекта  
Цель: упорядочить и облегчить работу на проекте

|  |  |
| --- | --- |
| Планирование тестирования | Отчетность |
| \* тест план \* график тестир \* матрица у-в  \* м-ца прослеживаемости  \* тестовый набор  \* тест сценарии  \* тест кейсы  \* чеклист + | \* отчет об ошибках +  \* отчеты о результате тестир |

**ПЛАНИРОВАНИЕ ТЕСТИРОВАНИЯ**

**I) Тест план** – документ, опис. весь объем работ по тестированию, начиная с описания тестируемых объектов, стратегии, расписания, критериев начала и окончания тестир, до необходимого в процессе работы оборудования, спец. знаний, оценки рисков с вариантами их разрешения

**Цель написания тест плана:**  
\* Продумать стратегию тестир ПО  
\* Описать процесс тестир на проекте и как он встраивается в процесс разработки  
\* Обесп информированность каждого члена к-ды об активностях QA к-ды, распределении обязанностей и зон ответственности  
\* Скорректировать ожидания заказчика от к-ды тестировщиков

**Стратегия тестир** – виды тестир, кот. будут проводиться над фичами и приложением, приоритеты, тест. документация и ее соотношение с видами тестир:  
для позитивного – тест-кейсы, для end to end – тест. сценарии и т.д.  
браузеры или поддерживаемые у-ва, на кот будем тестир, инструменты, к-мы будем польз. во t тестир, потенциальные проблемы, с кот. можем столкнуться в процессе тестир и план действий  
может также указ. обязанности члена к-ды тестир

**II) График тестир** – документ, опис. посл-сть выполнения активностей по тестир членами QA к-ды с указанием дат начала выполнения работ и их завершения

**Цель создания графика тестир:**  
\* Согласовать работу к-ды разработки и тестир  
\* Обесп информированность каждого члена к-ды о посл-сти задач а также о сроках их выполнения  
\* Обесп прозрачность процесса тестирования для заказчика  
\* Обесп возможность отслеживания отставаний с плана и влияния добавленных дополнительных задач к-де

**III) Матрица у-в** – документ используемый на проектах где разраб. мобильные приложения, кот. описывает конфигурацию у-в, где будет проводиться тестир  
составляется для тестир совместимости

**Цель создания матрицы у-в:**  
\* Определить оптимальное кол-во у-в, необх. для тестирования  
\* Убедиться что у-ва с различными версиями необх. оборудования участвуют в тестир  
\* Сосредоточить усилия тестировщика на у-вах, оптимально полно покрывающих требования проекта  
\* Обесп прозрачность тестир для заказчика

**IV) М-ца прослеживаемости** – документ, использ. для определения покрытия требования тестовой документации  
**Цель создания м-цы прослеж:**  
\* Обесп должное покрытие всех функц. и нефункц. требований тестами

**V) Тестовый набор** – документ, вмещающий в себя набор тестов/тестовых случаев/сценариев, собранные с опред. целью

**Цель создания тест. набора:**  
\* Сгруппировать случи по: видам тестир, уровням тестир, приоритету  
\* Обесп распределения объема тестир в к-де  
\* Обесп оценку трудозатрат тестировщика

По *приоритету* опред: smoke, ctirical path и extended testing

**VI) Тест сценарий** – документ, опис. посл-сть действий, кот. необходимо выполнить тестироващику с целью выявления дефектов в приложении (30-50 шагов, имитир работу реал. юзера)

**Виды тестир, для кот. пишут сценарии:**  
\* Исследовательское  
\* End-to-End  
\* Role-based

**VII) Тест-кейс** – документ, опис. посл-сть шагов и *ожидаемый* результат: Документ направл на проверку атомарных требований (тестир 1 треб: кнопка cancel – обычно 1 шаг)

**Виды тестир, для кот. пишут тест-кейсы:**\* Позитивное  
\* Негативное  
\* CRUD

**VIII) Чеклист** – документ, перечисляющий идеи для проверки. Документ, кот. очень поверхностно указ, что необх. проверить в приложении, но не указывает как это сделать

**Виды тестир, для кот. пишут чеклист:**  
\* Совместимости  
\* Инсталляции продукта  
\* Исследовательское  
\* CRUD  
\* Прерывания и т.д.

**ОТЧЕТНОСТЬ**

**I) Отчет об ошибке** – документ, опис. проблему, найденную в процессе тестир в приложении

**II) Отчет о результатах** – документ, опис результаты проведенного уровня или вида тестир, кот. дает полное понимание о кол-ве и серьезности найденных ошибок, кол-ве пройденных успешно и неуспешно тестов, а также о качестве самого приложения

**Жизненный цикл бага:**  
**1)** **opened (**есть в с-ме и не исправлен) – есть атрибут assigned: на какого разработчика зачислен, у него появл. нотификация, в завис. от приор и северити решает когда исправить  
как только взял в исправление -->  
**2)** **in process** – разработчик что-то сделал с багом -->  
**3)** **resolved** – с разными *резолюциями*:  
 *\* fixed* (разработчик согласился с багом, исправил)  
 *\* can’t reproduce* (разработчик не смог воспроизвести баг по шагам)  
 *\** *long fixed* (разработчик воспроизвел баг, но он расположен не в прил. а во Фреймворке)  
 *\* deferred* (отложено: поняли, что надо перелопатить все прил. и переделать все тестир)  
 *\* not a bug* (разработчик сказал: это не баг, это фича)  
**4.1) close** (баг исправлен и проверен: если *fixed*)  
**4.2) reopen** (делаем док-ва, видео: если *can’t reproduce*) -> п.1

# ЛК 12. Тест дизайн (ТД)

**ТД** – этап процесса тестир ПО, на кот. проектируются и создаются тестовые случаи (тест кейсы, чеклисты или тест сценарии), в соотв. с определенными ранее критериями качества и целями тестир.

**Процесс тестир ПО:**  
\* планирование и управление  
\* **анализ и проектирование  
\*** внедрение и реализация  
\* оценка критериев выхода и создание отчетов  
\* действия по завершению тестов

**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 = Ivan

**Категории техник ТД:**  
\* разработка тестов методом черного ящика  
\* метод создания тестов на основе опыта  
\* разработка тестов методом белого ящика

**МЕТОД ЧЕРНОГО ЯЩИКА:** **МЕТОД БЕЛОГО ЯЩИКА:**  
техника базируется на анализе техника базируется на анализе  
спецификаций или др. проектных внутр. структуры компонентов с-мы  
доков, опис. требования к с-ме

\\_\_ **МЕТОД НА ОСН. ОПЫТА** \_\_/

**Разработка тестов методом черного ящика:**  
\* для описания задач, кот. дб решены, программных продуктов или их компонентов, исп. модели – формальные или неформальные  
\* из этих моделей систематически выводятся тестовые сценарии

**Разработка тестов методом белого ящика:**  
\* тестовые сценарии выводятся на основе инфы о том, как спроектировано ПО (напр., 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 опред. тестовых сценариев исп. человеческие знания и опыт  
\* знания тестировщиков, разработчиков, юзеров и заинтересованных лиц о прогр. продукте, его использовании и окружении, явл. одним источником инфы  
\* знания о вероятных дефектах и их распределении явл. другим источником инфы

**ТД методом черного ящика:**  
\* эквивалентное разбиение  
\* анализ граничных значений  
\* тестирование таблицы решений  
\* тестир таблицы переходов  
\* тестир по сценариям использования

**Эквивалентное разбиение:**  
входные д-е для ПО или системы разбиваются на **группы**, от кот. ожидается сходное поведение, т.е. они должны обрабатываться аналогичным образом. Данные группы наз. *классами эквивалентности.*

**Система подбора кредита:**

Возраст до 18 => Кредит не разрешен //класс экив #1: от 1 до 17  
Возраст 18-65 => \* потребительский кредит  
 \* кредит на недвижимость //класс эквив #2: от 18 до 65  
 \* кредит на покупку авто  
Возраст 65+ => Потребительский кредит //класс эквив #3: от 66 до 99

**Анализ граничных значений:**  
Поведение на границах эквив. областей имеет наибольшие шансы быть некорректным, т.о. границы явл. потенциальным источником дефектов. Мин и макс значения сегмента явл. граничными значениями: **min -1, min, average, max, max +1**

**Система подбора кредита:**  
**Возраст до 18:**  
min -1: пустое поле «Возраст» > Сообщение об ошибке  
min: 1 > «Кредит не разрешен»  
average: 11 > «Кредит не разрешен»  
max: 17 > «Кредит не разрешен»  
max +1: 18 > «Потребительский кредит»

**Возраст 65+:**  
min -1: 65 > 3 кредита  
min: 66 > «Потребительский кредит»  
average: 80 > «Потребительский кредит»  
max: 99 > «Потребительский кредит»  
max +1: 100 > С-ма не позвол ввести больше 2 символов

**«ДАТА РОЖДЕНИЯ»**  
**Формат: дд.мм.гггг**  
день (значение):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | янв, март, май, июль, август, окт, дек | фев (високосный год) | фев (не високосный год) | апр, июнь, сент, ноя |
| 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 char | Empty | 1 char |
| Min | 1 char | @ | 2 char | - | 2 char |
| Average | 30 different chars | @ | 150 chars | - | 6 char |
| Max | 64 | @ | 243 chars | - | 11 char |
| Max +1 | 65 | @@ | 244 chars | - | 12 char |

**Тестирование таблицы решений:**  
Таблица решений содержит триггерные условия, обычно комбинации значений «Истина» и «Ложь» для всех входных условий, и рез. 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 тестир таблицы решений обычно явл. наличие хотя бы одного теста для каждой колонки, что обычно включает в себя покрытие всех комбинаций триггерных условий.

**Тестирование таблицы переходов:**  
Система может показывать различные отклики в завис. от текущих условий или предшествовавшей истории состояний. Данный метод позволяет тестировщику рассм. с-му с т.зр. ее 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.

Таблицы состояний демонстрирует связи между состояниями и входами и может подсказать возможные некорректные переходы.

**Система документооборота:**  
Созд. новый док --> временный док Temporary --> сохраненный док Saved --> просмотренный док Reviewed --> утвержд. док Approved --> выполненный док Processed --> заархивированный док Archived

|  |  |  |  |
| --- | --- | --- | --- |
| Тек. сост. | Событие | Действие | След. состояния |
| null | «Созд новый док» | create | temporary |
| null | «Созд новый док» | discard | null |
| temporary | Заполнение дока | save | saved |
| temporary | Заполнение дока | cancel | null |
| saved | Просмотр дока | review | reviewed |
| saved | Просмотр дока | reject | rejected to creator |
| reviewed | Утверждение дока | approve | approved |
| reviewed | Утверждение дока | reject | archived |
| approved | Вып действий по доку | process | processed |
| processed | Архивация дока | archive | archived |

**Сценарий испо-ния** (use case) опис взаимодействие м/участниками (вкл юзеров и систему) приводящие к полезным рез-там для заказчика или юзера с-мы. Сценарии исп. мб опис на уровне абстракций (бизнес-сценарий исп., уровень бизнес-процессов, не связ. с технологией) или на сист. уровне (сценарий исп системы на уровне сист. функционала)

**Тест дизайн методом белого ящика:**

* *Компонентный уровень:*  
  структура компонента ПО, т.е. операторы, альтернативы, ветви или определенные пути
* *Интеграционный уровень:*  
  структура мб предст деревом вызовов (диаграмма, в кот. модули вызывают другие модули)
* *Системный уровень:*  
  структура м предст собой структуру меню, бизнес-процессов или же схему веб-страницы

**Техники тест дизайна:**

* Тестирование покрытия ~~операторов~~ (statement coverage testing)
* Тестирование покрытия альтернатив (decision coverage testing)
* Тестирование покрытия условий (condition coverage testing)

**1. Тестирование покрытия стэйтментов:**  
Техника тестир, кот. включает в себя выполнение всех операторов хотя бы единожды. Полученная метрика позволяет высчитать кол-во операторов, кот. были выполнены и кот. вообще есть в коде.

**2. Тестирование покрытия альтернатив:**  
Техника тестир, кот. связана с тестир ветвей (branches) (напр., вариантов True, False для оператора IF). В методе тестир альтернатив тестовые сценарии созд. для выполнения опред. рез-тов альтернатив. Ветви исходят из точек альтернатив в программном коде и показ. передачу управления различным участкам кода.

*!Тестирование альтернатив – это вид тестир потока управления, так как оно опис прохождение опред. потока через точки альтернативы*

*Покрытие альтернатив более строгое, чем покрытие стейтментов (если 100% покрытие альтерн -> значит 100% операторов, но не наоборот)!*

**3. Тестирование покрытия условий:**Покрытие условий (condition) и покрытие множественных условий (multiple condition) – это более высокий уровень покрытия структуры кода после покрытия альтернатив. Техника предполагает покрытие всех условий, кот. могут повлиять на рез. принятия решения.

**Тест дизайн на основе опыта:**

* Предположения об ошибках (error guessing technique)
* Исследовательское тестир (exploratory testing)

**Предположения об ошибках:**  
Организационным подходом к предположению об ошибках явл. создание списка возможных дефектов и разработка тестов для атаки этих дефектов. Данный подход наз. *атакой*, или *атакой на недочеты*. Списки дефектов и отказов мб созданы на основе опыта, доступной инфы о дефектах и отказах и общего представления о том, почему ПО может отказать.

# ЛК 13. Тестовые случаи

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

**Цель написания test case-ов:**

* Подготовиться к тестированию (осн. цель)
* Детально описать шаги тестир и ожидаемый результат
* Задокументировать требования
* Облегчить передачу знаний по проекту
* Подготовиться к автоматизации тестирования

**Атрибуты test case-а: (обяз. поля)**

* **ID** – tc№
* **Краткое название тест случая (Summary, Title)** – уник среди tc, сост т. из существительных
* **Цель тест случая (Goal, Aim, Description)** – что проверяем и при каких усл (1 предл: Verify that…..)
* **Предусловия (Precondition)** – нумерованный список шагов чтобы добраться до компонента (в этом списке ожид. рез прописывать не надо)
* **Шаги (Steps)** – опис т. д-вия, напр на проверку выбранного требования)
* **Ожидаемый результат (Expected Result)** – пропис к каждому шагу, дб подробным и однозначным
* Постусловия (Postcondition) – д-вия, чтобы вернуть с-му в исх сост
* **Статус (Status)** – есть всегда:   
  \* tc не пройден (not run)  
  \* успешно пройден (passed) : ожид рез = актуальный рез  
  \* неудачно пройден (failed) : ожид рез != актуальный рез  
  \* заблокирован (blocked) : не мб вып шаг либо предусловие tc
* Уровень (Level) – выставл в соотв с уровнем тестир, для кот. он предназначен:  
  \* компонентный  
  \* интеграционный  
  \* системный  
  \* приемочный
* Приоритет (Priority)   
  \* высокий : д пройти в 1 очередь   
  \* средний : 2 очередь  
  \* низкий : 3 очередь
* Автор (Author) – в с-мах tc management выставляется авто-
* Комментарии (Notes, Comments) –

То, что жирным, обязательно  
Делить шаги так, чтобы параллельно был с таким же номером Expected Result  
(1. ост пустым И нажать Логин || 1. error message)

**Правила написания ТС:**

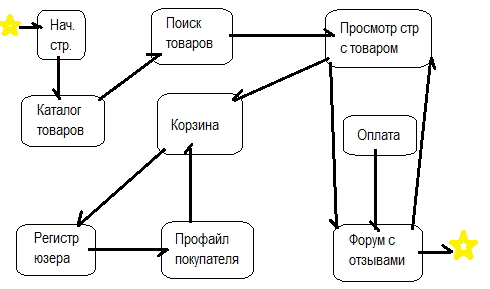
* 1 требование = 1 ТС
* Кол-во шагов не больше 7
* В Steps/Expected Result – только то, что относится к цели ТС
* Шаги, кот. не относятся к цели ТС – это либо Pre- либо Postconditions
* Минимум кликов – максимум результата
* Тест кейсы д идти последовательно, составляя сценарий
* Тест кейсы дб разделены на смысловые части
* Как можно меньше ссылок на сторонние доки
* Без картинок

# ЛК 14. Тестовые сценарии

**Тест. сценарии** – формально описанный алгоритм тестир программы, специально созд. для опред. возникновения в проге опред. ситуации, опред. выходных д-х

|  |  |
| --- | --- |
| Тест. случай | Тест. сценарий |
| 1. провер. одно требования 2. ппровер. функциональность и внешний вид 3. содерж. не больше 5-7шагов 4. каждый шаг содерж. ожид. рез. | 1. провер. много требований 2. провер. только функц-сть 3. «чем больше шагов, тем лучше» 4. ожид. рез. не прописывается |

**Цель написания тест. сценариев:**  
подготовиться к проведению след. видов тестир:  
\* end-to-end  
\* explorary testing  
\* role-based testing

**1) End-to-end тестир:**  


**2) Role-based тестир:**  
*Content Manager:* 1. Заходит в admin часть прилож  
 2. Доб. новый товар в каталог  
 3. Заполняет техн. описание товара  
 4. Переводит товар в статус «Новый»

*Marketing Manager:* 5. Откр товар со статусом «Новый»  
 6. Выставл цену товара  
 7. Созд. с-му скидок на товар  
 8. Переводит в статус «Готово»

*Buyer:*  9. Откр прилож и заходит под своим логином/паролем  
 10. На нач. страинце видит новый товар  
 11. Оформл покупку  
 12. Получ. скидку согласно опред. выше с-ме  
 13. Оплач. товар  
 14. Выходит из прилож

**3) Исследов. тестир:**  
\* The GuideBook Tour (Путеш по путеводителю)  
\* The Money Tour (Денежное путеш)  
\* The LandMark Tour (Путеш по достопримеч)  
\* The Back Alley Tour (Путеш по переулкам)  
\* The Supporting Actor Tour (Путеш суфлера)  
\* The Linely Businessmen Tour (Путеш одинокого бизнесмена)

**Атрибуты тест. сценария:**  
\* Идентификатор (ID)  
\* Краткое название (Summary, Title)  
\* Описание сценария (Description) 1-5 предл  
\* Участники (Primary Actors) роли (юзеры), кот. фигурир в ТС  
\* Предусловия (Precondition) подгот. с-му к вып. сценария  
\* Осн. сценарий (Basic Flow)  
\* Альтернатив. сценарий (Alternative Flow)  
\* Исключения (Exceptional Flow)  
\* Статус (Status) как у ТК  
\* Приоритет (Priority) как у ТК (выс, ср, низ)

ТС счит. **failed** если в процессе прохождения были найдены баги, с-ма либо противореч треб либо здравой логике  
С-ма не явл. участником, никогда не прописываем в актерах

**Basic Flow:**  
посл-сть шагов или д-вий, напр. для достижения цели. Это прямой путь, при кот. использ. значение по умолч и выбир. д-вия, для кот. с-ма непосредственно предназначена  
(всегда позитив. тесты)

**Alternative Flow:**  
посл-сть шагов или д-вий, ведущая к достиж. цели юзера, но кот. подразумевает отклонение от направления по умолч  
(всегда позит тесты)

**Exceptional Flow:**  
посл-сть шагов или д-вий, кот. препятствует достиж. цели и кот. подразумевает отклонение от правильной работы прилож  
(всегда негат тесты)

3 юзера:  
\* незарег, зарег и logged-in user

# ЛК 15. Отчеты о результатах тестир

**Отчет о рез. тестир** – документ, обобщающий разультаты работы по тестир и содерж инфу, дост. для соотнесения текущей ситуации с тест-планом и принятия необх. управленческих решений

**Цель написания:**  
\* предост статистику по кол-ву проверок и рез-там их выполнения  
\* предост статистику по кол-ву и серьезности найденных дефектов в прилож  
\* повысить осведомленность команды о кач-ве текущей версии прилож  
\* обратить внимание к-ды и владельцев бизнеса на области, кот. нуждаются в исправлении дефектов  
\* дать рекомендации относительно выпуска или наоборот задержки выпуска версии в релиз

**Структура отчета:**

1. Уровень/вид проведенного тестир  
2. Кол-во пройденных проверок (test cases/test scenarios/checkpoints)  
 \* общее кол-во  
 \* кол-во успешно пройденных (passed)  
 \* кол-во неудачных (failed)  
 \* кол-во заблокированных проверок (blocked)  
3. Кол-во найденных багов  
 \* общее кол-во  
 \* кол-во critical багов  
 \* кол-во major багов  
 \* кол-во minor багов  
 \* кол-во trivial багов  
4. Области/компоненты, кот. наиб. подвержены багами (более 75% проверок failed)  
5. Проблемы, с кот. столкнулись в процессе вып задачи

В отчете также мб указаны номера билдов, на кот. проводилось тестир (в самом верху) и члены команды, кот. проводили данные виды тестир

# ЛК 16. Тест план

**Тест стратегия** – офиц. док, опис. методологию тестир, принятую в компании.  
Одна и та же организация м. иметь разные стратегии для разных продуктов, рзных циклов разработки, разных уровней риска.

**Виды тест стратегий:**

* Аналитические стратегии  
  - тестир, осн. на рисках (risk-based testing)  
  - тестир, осн. на требованиях (requirements-based testing)
* Стратегии, осн на моделях  
  - тестир, осн на моделях использ приложения
* Методические стратегии  
  - тестир по общепринятым стандартам (ISO 25010:2011)  
  - тестир по стандартам, принятым в компании
* Стратегии, соотв процессуальным нормам  
  - тестир по стандартам HIPPA, GDPR и т.д.
* Реактивные стратегии  
  - исслед тестир
* Консультативные стратегии  
  - тестир, осн на сценариях и д-х, предост заказчиком ПО
* Стратегии, искл регрессионное (повторное) тестир  
  - широкое использ автоматизации для любых повтор. тестов

**Отличие:**   
*Методические стратегии* ориент на результат, насколько качественное вышло ПО  
*Стратегии соотв процесс. нормам* - стратегии, ориент на процессы, насколько наши процессы разработки соотв. стандартам

Разные стратегии мб объединены в одну для того, чтобы обеспечить макс достижение целей в тестировании, принятых в компании

**Тест стратегия вкл:**

* Описание процесса интеграции тестир в процессе разработки
* Техники тест дизайна
* Методы и виды тестир
* Уровни тестир
* Обяз и необяз стандарты, к-рым д. соотв ПО
* Критерии начала и окончания тестир
* Метрики, собираемые в процессе тестир
* Инструменты, исп в тестир
* Окружение, где проходит тестир
* Процесс контроля качества и его метрики
* Дефект менеджмент
* Роли и обяз-сти членов команды тестир

**Проектный тест план:**  
офиц. док, опис стратегию тестир, принятую на конкр. проекте. Если разраб неск приложений, Мастер Тест План должен опис пересечение стратегий тестир всех приложений

* Опис, что будет тестироваться, а что нет
* Качеств. хар-ки, кот. будут оцениваться, и не будут
* График тестир и бюджет
* Циклы тестир и их соотнош с релизным планом проекта
* Взаимоотношения между командой разработки и тестир, рез-ты работы (deliverables)
* Критерии начала и оконч тестир
* Риски в тестир
* Общее руководство процессом тестир на проекте
* Обязанности тестировщиков в рамках тестир каждого продукта
* Тестовые д-е и выходные д-е после тестов

**Level тест план**  
офиц. док, опис. стратегию тестир, принятую для конкр. уровня тестир или вида тестир. Состоит из графика тестир, задач, дат оконч работ, и других моментов, кот. отличаюются или не описаны в Мастер Тест Плане

# ЛК 17. Отчеты об ошибках

**Отчет об ошибках** – док, кот. описывает и приоритизир. обнаруженный дефект, а также содействует его устранению.

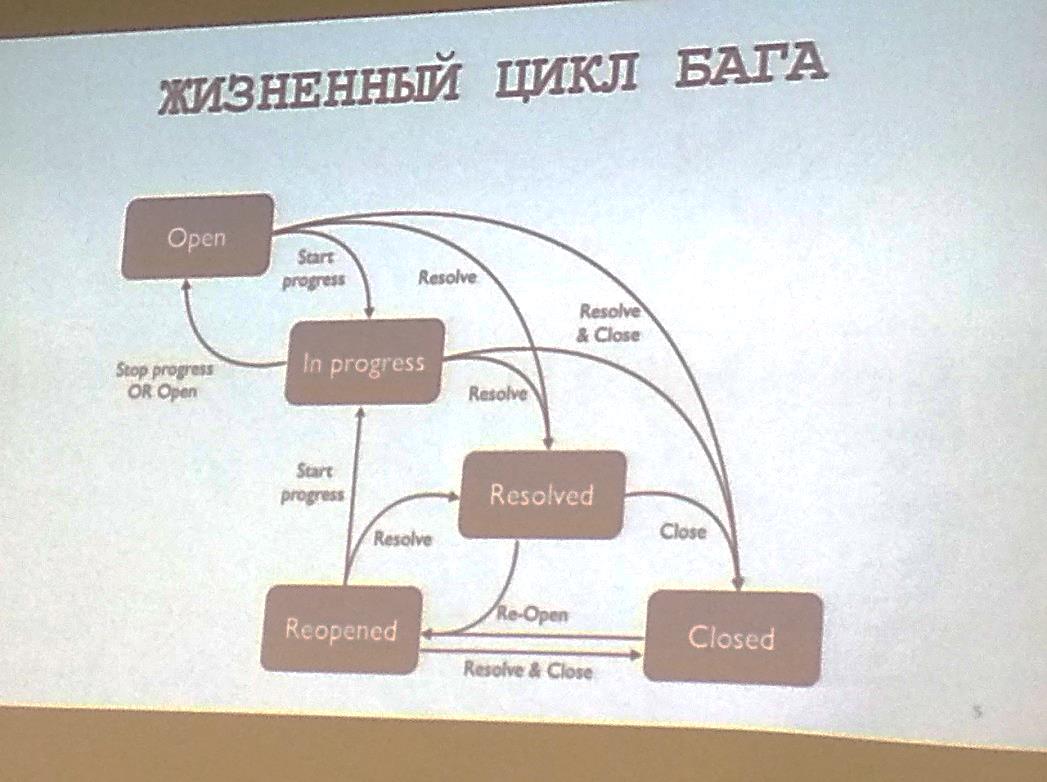
**Дефект** – любое отклонение факт. результата от ожиданий пользователя, сформир. на основе требований, спецификаций, иной документации или опыта и здравого смысла

**Цель отчета об ошибках:**

* **Предоставить инфу о проблеме:**  
  уведомить проектную к-ду и иных заинтересованных лиц о наличии проблемы, описать суть проблемы
* **Приоритезировать проблему:**  
  опред степень опасности и желаемые сроки ее устранения
* **Содейств устранению проблемы:**  
  предост необх. подробности для понимания сути случившегося, а также анализ причин возник проблемы и рекомендации по исправл ситуации

**Жизн. цикл бага:**

*разработчик не имеет права закрывать дефект! (только тестировщик)*



**Резолюции отчета об ошибке:**

* исправлен (fixed)
* невозможно воспроизвести (can’t reproduce)
* не будет исправлен (won’t fixe)
* работает, как задумано (as designed)
* дубликат (duplicate)
* отклонен (deferred)

**Атрибуты отчета об ошибках:**

* **ID**
* **Краткое описание**
* **Описание : Шаги воспроизв, Актуальный, Ожидаемый рез.**
* Воспроизводимость
* **Важность (Severity)**
* **Срочность (Priority)**
* возможность обойти
* комментарий
* **среда воспроизв**
* компонент
* тег
* **ответственный**
* приложения к багу
* срок исправления бага
* версия билда, где был обнаружен баг
* версия, в кот. дб исправлен баг
* **создатель отчета об ошибке (авто-)**
* оценка трудозатрат на исправления бага

|  |  |
| --- | --- |
| Важность бага | Приоритет бага |
| \* блокирующий \* критичный \* значительный \* незначительный \* тривиальный | \* высокий (на гл. странице) \* средний \* низкий (вряд ли юзер туда дойдет) |

**Свойства качественных отчетов:**

* тщательной заполнение всех полей точной и корректной инфой
* отсутствие «размытых» формулировок («иногда неправильно работает», «UI сломан», «кнопка Export не работает»)
* вся инфа, особенно Ожидаемый результат, корректна
* баг заведен на функциональность, кот. разработана (неразработанный функционал за ошибку не считается)
* отсутствует жаргонная лексика
* отсутствие критики кого-то из участников к-ды («Ну каким дураком надо быть, чтобы такое сделать?!»)
* важность и приоритет бага корректно выставлены
* к отчету прикреплены все возможные доки, кот. облегчат понимание причины дефекта
* отсутствуют грамматические и семантические ошибки («Not keyboard in parameters accepting values»)
* использован правильный технический язык
* краткое описание бага должно следовать правилу WWW
* в отчете дб описани только 1 дефект
* оформление соотв. соглашению о названиях, принятом на проекте

**Инструмент для отчетов о багах** – с-ма для созд, хран 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 соотв. сотрудникам
* Организация взаимосвязей между отчетами о дефектах, тест-кейсами, требованиями, анализ таких связей с возможностью формирования рекомендаций
* Подготовка инфы для включения в отчет о результатах тестир
* Интеграция с с-ми управления проектами