Лекция №5 Исследовательское тестирование

### **Исследовательское тестирование**

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

“A Tutorial in Exploratory Testing” Cem Kaner, 2008

Более простое определение исследовательского тестирования — это разработка и выполнения тестов в одно и то же время. Что является противоположностью сценарного подхода (с его предопределенными процедурами тестирования, неважно ручными или автоматизированными). Исследовательские тесты, в отличие от сценарных тестов, не определены заранее и не выполняются в точном соответствии с планом. Но при этом, даже в свободной форме поисковой сессии тест будет включать в себя ограничения состоящие в том, какую часть продукта тестировать или какую стратегию использовать.

**Цели исследовательского тестирования:**

* Проверить работу системы с точки зрения 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
   * уточнение тестируемого компонента;
   * уточнение цели тестирования (миссии);
   * уточнение окружения (платформы/браузеры/устройства/т.д.)
   * уточнение продолжительности сессии
   * 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, какие баги при этом были найдены, с какими трудностями в приложении столкнулись.

Цель тестирования (**миссия**) - это описание цели сессии исследовательского тестирования: на что тестировщик планирует обращать внимание, какой аспект фичи/приложения планирует проверять.

**Продолжительность сессий:**

* “Короткая” (60 мин);
* “Нормальная” (90 мин);
* “Длинная” (120 мин).

**Документация в исследовательском тестировании:**

* Чеклист (общий, краткий список с проверками);
* Тест Чартер (подробный документ с деталями тест-сессии);
* Записи шагов тестировщика в приложении (с помощью ПО типа Jira Capture, Session tester, qTrace и другие);
* Отчеты об ошибках;
* Логи приложения (если уместно).

#### **Тест Чартер: структура**

* ID;
* Чартер (название чартера напр. "Тур, испорченный из-за дождя", "Тест критического пути формы входа");
* Миссия (цель сессии);
* Области (список компонентов/модулей, которые будут затронуты тестированием);
* Окружение и URL (ссылка на тестируемое приложение, ОС/браузеры/устройства, где будет проходить тестирование);
* Дата и время старта;
* Имя тестировщика;
* Длительность (короткая, нормальная, длинная и количество минут);
* Записки тестирования (список проверок, что планируется проверять);
* Файлы данных (если требуется);
* Отчеты об ошибках (названия (title) багов и их ID);
* Проблемы (описание трудностей при тестировании, если были);
* Время на подготовку к тестированию (например, установку нового билда на мобильное устройство, необязательное поле);
* Тест дизайн и Тестирование (процент времени, потраченный на продумывание проверок и на само тестирование, необязательные поля).

### **Туры исследовательского тестирования**

Чтобы систематизировать [исследовательское тестирование](https://dist.belstu.by/mod/assign/view.php?id=57436) можно использовать идею **туров**.  Туры – это идеи и инструкции по исследованию программного продукта, объединенные определённой общей темой или целью. Туры, как правило, ограничены по времени – длительность тестовой сессии не должна превышать 4 часа.

Идею туров развивали в своих работах Канер, Бах, Хендриксон, Болтон, Кохл и другие.

Джеймс Виттакер, хоть и не придумал саму идею туров, но предложил свой подход к исследовательскому тестированию с использованием туров и в своей книге “Exploratory Software Testing” в доступной форме озвучил идею туров и описал сами туры.

Тур – это своего рода план тестирования, он отражает основные цели и задачи, на которых будет сконцентрировано внимание тестировщика во время сессии исследовательского тестирования. При этом Виттакер использует метафору, что тестировщик – это турист, а тестируемое приложение – это город. Обычно у туриста (тестировщика) мало времени, поэтому он выполняет конкретную задачу в рамках выбранного тура, ни на что другое не отвлекаясь. Город (ПО) разбит на  районы: деловой центр, исторический район, район развлечений, туристический район, район отелей, неблагополучный район.

#### **Туры по бизнес-центру (Tours of the Business District)**

Ассоциация с районом в туризме: Это район, где жители города “делают деньги”, выполняют работу. Это район, в котором туристам, часто неинтересно. Он заполнен банками, офисными зданиями. По утрам и вечерам там можно попасть в многочасовую пробку.

Сравнение с тестированием приложения: фичи, которые “делают бизнес” – они появляются в промо-материалах, ради них пользователь приобретает приложение.

**Тур по путеводителю (Guidebook Tour)**

Ассоциация в туризме: турист, строго использующий путеводитель в своём путешествии.

Сравнение с тестированием приложения: пользователь, строго идущий по шагам, описанным в справочной информации, специально вызывающий подсказки и вычитывающий их.

Типичные баги:

* Несоответствия описанных шагов в справке работе приложения
* Несоответствия скриншотов в справке и самого приложения
* Ошибки (от орфографических до логических) в справке
* Ошибки удобства использования справки: длинная или короткая, непонятная или примитивная, не отвечающая на те вопросы, которые предполагают прочитать пользователи
* Ошибки удобства использования приложения

**Денежный тур (Money Tour)**

Ассоциация в туризме: путешествие по тем местам, ради которых туристы хотели приехать в город. Если бы вдруг не стало талисмана Парижа, Эйфелевой башни, то приток туристов в Париж резко бы уменьшился.

Сравнение с тестированием приложения: проверка на соответствие информации в рекламных материалах и в приложении: тестирование тех фич, которые продают приложение, фич, которые используют менеджеры по продажам на демонстрациях продукта. Контроль качества этих фич и информирование маркетологов о всевозможных проблемах, которые могут повлиять на рекламу и продажи. О фичах обычно можно узнать из демо-презентаций, страницы продукта на сайте, рекламных брошюр, рекламных баннеров и т.п.

Типичные баги:

* Напрочь отсутствие в приложении некоторых описанных в промо-материалах возможностей
* Несоответствие информации в рекламных материалах и работы самого приложения
* Функциональные ошибки в работе приложения, влияющие на демонстрацию
* Устаревшие скриншоты приложений
* Лингвистические ошибки (от орфографических до логических)

**Тур по ориентирам (Landmark Tour)**

Ассоциация в туризме: путешествие по заранее выбранным местам в намеченном, но не продуманном детально, порядке, как если бы он путешествовал с путеводителем. Эти выбранные места и являются ориентирами для туриста – способ и всё посмотреть, и не заблудиться.

Сравнение с тестированием приложения: тестировщик определяет список фич приложения, которые ему необходимо проверить, намечает примерный путь прохождения этих фич, и далее начинает проверку, переходя от проверки одной фичи к другой. При этом сам список служит ему ориентиром и напоминанием о том, чтобы он не сбивался с пути. Это тур не только о списке фич, но также и о выбранной последовательности их прохождения.

Типичные баги: блокирующие, критические проблемы в работе приложения.

**Тур интеллектуала (The Intellectual Tour)**

Ассоциация в туризме: представьте туристическую группу и в ней туриста, который отлично разбирается в истории, познакомился с обычаями местности и перечитал много информации о городе. И вот этот турист начинает задавать гиду-хитрецу, просто заучившему текст экскурсии, ну очень умные вопросы!

Сравнение с тестированием приложения: задавать приложению очень умные и неожиданные вопросы. А именно, при работе приложения задумываться о сложных, но реалистичных сценариях, обычно связанных с объемом данных, скоростью выполнения операции, осознанном вызове ошибок:

* в проигрывателе прослушать аудиофайл длительностью в 10 часов
* в текстовом редакторе создать объемный файл
* указать длинное имя файла
* при работе мобильного приложения вызывать телефонный звонок
* выполнять какое-то действие очень быстро
* на форме заполнить все поля ошибочными данными

Типичные баги:

* Креши приложения при выходе за граничные значения
* Автоматические прерывания текущих процессов, изменение состояние приложения
* Недоработки в логике
* Низкая производительность, медленная работа приложения
* Потеря данных
* Примитивные ошибки в удобстве использования (например, в файловом менеджере мало места отведено под название документа и нет возможности его просмотреть другим способом)

**Тур службы доставки (FedEx Tour)**

Ассоциация в туризме: большинство из нас пользуется службами доставки. Мы формируем посылку, относим её на почту и ожидаем, что она будет доставлена до места назначения. В процессе, на посылке могут появляться дополнительные метки, но ожидается, что посылка всё равно найдет адресата.

Сравнение с тестированием приложения: пользователь довольно часто вводит какие-то данные (ФИО, адрес доставки, имя документа, сумму транзакции и т.д.). Внутри приложения эти данные могут использоваться в различных операциях, отображаться на экране, изменяться и когда-то даже удаляться. Задача тестировщика – проследить путь, по которому идут данные. А именно, определить места, где происходит их ввод, где они используются, изменяются, отображаются и проверить все эти места, выполняя соответствующие операции.

Типичные баги: совершенно разные – от критических функциональных до багов в удобстве использования, интерфейсе, локализации (поддержка форматов разных стран).

**Тур после работы, внеурочный тур (The After-Hours Tour)**

Ассоциация в туризме: после работы все спешат домой или на встречу с друзьями, и, как результат, в бизнес-центре начинаются часы-пик. Туристы чаще всего пытаются остаться вдали от этих мест. А вот тестировщики нет!

Сравнение с тестированием приложения: то, что пользователь перестает использовать приложение, не обозначает, что приложение останавливает работу. Часто приложение бэкапит данные, архивирует, получает обновления и т.д. Тестировщик должен проследить за этими операциями.

Типичные баги:

* проблемы неэффективного использования ресурсов (нерациональное использование трафика, места на диске, процессора, утечки памяти и т.п.)
* функциональные ошибки иногда приводящие к остановке работы приложения

**Тур мусоросборщика (The Garbage Collector’s Tour)**

Ассоциация в туризме: каким бы ни было место путешествия, везде есть мусоросборщики. Наблюдая за ними, понимаешь, что эти ребята знают разных жителей района и сам район лучше, чем даже их соседи, или полиция, потому что они переходят с улицы на улицу, останавливаются у каждого дома и, в итоге, знают каждый камень на дороге. Они не задерживаются у дома надолго, они методично переходят от точки к точке.

Сравнение с тестированием приложения: для тестировщика этот тур похож на выборочную проверку, в процессе которой он по кратчайшему пути должен методично переходить с экрана на экран, вызывать диалог за диалогом и проверять какую-то простую примитивную вещь, не углубляясь особо в детали. Например, наличие и правильность заголовков на всех окнах, зеленый цвет кнопки сохранения везде, где она есть, наличие метки с количеством комментариев к статье везде, где эта статья как-то упоминается и т.д.

Типичные баги: в зависимости от выбранного сценария. Если он связан с текстом, то баги, в основном, лингвистические, с функционалом – функциональные и т.д.

#### **Туры по историческому району (Tours Through the Historical District)**

Ассоциация с районом в туризме: это старые районы города со старыми зданиями, или местами, имеющими историческую ценность.

Сравнение с тестированием приложения: это места, связанные с предыдущей версий, со старым функционалом или кодом.

**Тур по нерекомендуемым местам (The Bad-Neighborhood Tour)**

Ассоциация в туризме: в каждой местности есть места, которые туристам советуют избегать.

Сравнение с тестированием приложения: в приложении – это фичи или места в коде с наибольшим скоплением багов. Про эти места вы можете знать интуитивно, а можете воспользоваться багтрекером. Как только вы нашли один баг, знайте, что, скорее всего, рядом есть и другой; он может находиться в том же функционале, а может и в соседнем. Задача тестировщика состоит в том, чтобы пройти по местам. Связано это с тем, что баги имеют свойство скапливаться в одном месте.

Тур также применим и после исправления багов. Пройдитесь взаимосвязанным областям.

Типичные баги: функциональные.

**Музейный тур (The Museum Tour)**

Ассоциация в туризме: фаворитами туристов являются музеи, отображающие античные, средневековые времена.

Сравнение с тестированием приложения: код тоже может быть “античным”. Это тот код, который давно не изменялся. Такой код, попадая в новую среду, может вообще не работать, или, подвергаясь внешнее ревизии, может оказаться непригодным. Тестировщики могут найти такой код по метке даты изменения в репозитории.

Например, если создают для iOS новый, но схожий с другим проект, то могут скопировать лишний код или ресурсные файлы, которые при проверке Apple ревьюверами может привести к отклонению приложения.

Типичные баги:

* креши
* функциональные ошибки
* несоответствие стандартам или гайдлайнам
* увеличение размера приложения

**Тур предыдущей версии (The Prior Version Tour)**

Ассоциация в туризме: после реставрации здания должны оставаться привлекательными для туристов, т.к. обновление до неузнаваемости может уменьшить поток туристов.

Сравнение с тестированием приложения: после обновления приложения пользователь должен легко и быстро адаптироваться к новому функционалу, или новым путям использования приложения. Особенно это важно проверять, когда вырезается какой-то функционал, изменяется интерфейс или даже исправляется очень старый баг, который некоторые пользователи могли использовать как фичу.

**Типичные баги:**

* ошибки юзабилити
* функциональные ошибки, связанные с утратой функциональности, потерей данных, ошибками в логике

#### **Туры по району развлечений (Tours Through the Entertainment District)**

Ассоциация с районом в туризме: туристам нужны места, где можно отдохнуть, оторваться от плотного графика переездов и осмотра города.

Сравнение с тестированием приложения: в большинстве приложений есть места, где можно отвлечься от основной бизнес-задачи (сценария использования). Например, настроить приложение под свои особенные нужды, или навести “красоту” изменяя шрифт и цвет текста и т.п.

**Тур актера второго плана (The Supporting Actor Tour)**

Ассоциация в туризме: гиды чаще всего рассказывают о самых популярных местах, а то прекрасное и интересное, но не популярное, что находится рядом с этими известными местами, опускают.

Сравнение с тестированием приложения: выполняя этот тур, используйте, изучайте, проверяйте не ключевой функционал, а тот, что его окружает – не главный, не столь заметный, но такой, который найдёт своего пользователя.

Типичные баги: разные, но больше интерфейс, лингвистика или некритичные функциональные ошибки.

**Тур по темным переулкам (The Back Alley Tour)**

Ассоциация в туризме: не все туристы посещают только самые популярные места, некоторые заглядывают и в темные переулки, а многие хотя бы раз заходят в туалет… Эти места как бы “за сценой” основного действа, но и о них нельзя забывать.

Сравнение с тестированием приложения: это путешествие по фичам, которые маловероятно, что будут использоваться, по фичам, наименее привлекательным для пользователей. Если у тестировщика есть статистика по использованию приложения, то в этом туре он может следовать по фичам из самых нижних позиций рейтинга.

Типичные баги:

* некритичные функциональные ошибки
* определение мест и фич, назначение которых пользователь вообще не понимает без подсказки или справки
* ошибки юзабилити: примеры странного, нелогичного поведения
* ошибки лингвистики
* ошибки интерфейса

**Тур любителя ночной жизни (The All-Nighter Tour or Clubbing Tour)**

Ассоциация в туризме: некоторые туристы живут в режиме non-stop. Днем – пляжи, ночью – дискотеки. И всегда есть еще одно место, куда хочется заглянуть.

Сравнение с тестированием приложения: сколько сможет выдержать ваше приложение без перезагрузки? Никогда не перезапускайте приложение! Используйте одну и ту же фичу постоянно, повторяйте одно и то же действие раз за разом. Примеры: открыть файл и никогда его не закрывать, соединиться с сервером и никогда не разрывать соединение, запустить приложение и никогда его не перезапускать.

Типичные баги:

* проблемы производительности и эффективности такие как утечки памяти, медленная скорость работы
* потеря данных
* функциональные ошибки

#### **Туры по туристическому району (Tours Through the Tourist District)**

Ассоциация с районом в туризме: в каждом городе есть места с наибольшей концентрацией туристов. Эти места заполнены сувенирными магазинами, ресторанами и т.п.

Сравнение с тестированием приложения: быстрые проверки, с целью просто пробежаться по функциям. Мол, “я здесь был”.

**Тур коллекционера (The Collector’s Tour)**

Ассоциация в туризме: многие туристы хотят оставить память о посещенных местах, покупая магнитики, открытки и прочие сувениры.

Сравнение с тестированием приложения: собирайте всевозможные артефакты вашего приложения – то, что пользователь может сохранить себе, оставить “на память”; это всё, что приложение “отдает” пользователю. Например, в браузере – это история, закладки, посещенные страницы, в файловом менеджере – файлы разных форматов, папки всевозможные и т.д.

Типичные баги: в основном, некритичные функциональные ошибки

**Тур одинокого бизнесмена (The Lonely Businessman Tour)**

Ассоциация в туризме: некоторые бизнесмены, путешествуя по разным городам, по сути видят только аэропорт, офис и отель. Некоторые адаптируются: бронируют отель как можно дальше от офиса и, таким образом, по дороге из офиса в отель, или из отеля в офис, пытаются увидеть как можно больше.

Сравнение с тестированием приложения: доберитесь до фичи, которая находится глубже всего, дальше всего от старта приложения. Или же выберите какую-то фичу и доберитесь до неё самым длинным путем, какой сможете придумать.

Типичные баги: разные – от функциональных (чаще некритических), до багов в удобстве использования, интерфейсе, локализации, производительности или в не полноте справки.

**Тур супермодели (The Supermodel Tour)**

Ассоциация в туризме: далеко не всякое путешествие несет некую материальную ценность. Иногда мы останавливаемся просто полюбоваться красотами. А иногда мы просто едем куда-то “просто посмотреть”.

Сравнение с тестированием приложения: этот тур о внешнем интерфейсе приложения: красив ли, привлекателен ли, правильно ли используются цвета, нет ли лишних артефактов в интерфейсе, интересная и быстрая ли анимация, соответствует ли интерфейс принятым стандартам, ожиданиям пользователя.

Приложение может содержать много багов, но и как супермодель на подиуме, оно будет смотреться великолепно!

Типичные баги: проблемы интерфейса и удобства использования.

**Тур шопоголика (The TOGOF Tour – Test One Get One Free)**

Ассоциация в туризме: все туристы ходят по магазинам, а некоторые и едут специально в туры “за покупками”, где частенько встречают таблички “Купи одну, получи одну бесплатно” (Buy One Get One FREE (BOGOF)). А при серьезных распродажах за какую-то вещь ещё и побороться приходится. 🙁

Сравнение с тестированием приложения: необходимо создать ситуацию, в которой несколько копий одного и того же приложения используют один и тот же объект, одну и ту же фичу в один момент времени. Например, запустить несколько приложений и обращаться к одному файлу или разделу виртуальной памяти; залогиниться несколькими пользователями в разные приложения; удалять и редактировать одновременно одну и ту же статью; по одной сети отправлять на сервер конфликтующие данные в один момент времени и т.д.

Почему TOGOF? Найдя баг в одной копии приложения, вы находите его и в другой. Бесплатно 🙂

Типичные баги:

* недоработки в логике, приводящие как к некритичным, так и к критичным багам
* непонятные для пользователя сообщения об ошибках (формата “error 784”)
* проблемы производительности

**Тур по шотландским пабам (The Scottish Pub Tour)**

Ассоциация в туризме: не все интересные, стоящие, важные места можно увидеть по путеводителям, и с помощью гидов. Чтобы увидеть некоторые места нужно заблудиться в городе, побродить по переулкам, или поспрашивать местных жителей. И лучшее место, где можно поболтать с закоренелыми местными – это пабы. На самом деле и сами пабы часто являются особой достопримечательностью, про которую не пишут рекламные брошюры, но которую стоит увидеть своими глазами.

Сравнение с тестированием приложения: на форумах, в блогах, при общении с пользователями, да и просто блуждая по приложению можно найти неожиданные для себя, как тестировщика, фичи, но те, которые полюбились настоящему ценителю-пользователю. Этот тур наиболее хорош для очень больших приложений. Его задача состоит не только в том, чтобы проверить, но и в том, чтобы глубже познакомиться с приложением (в идеале, посредством общения с пользователем).

Типичные баги: совершенно разные.

#### **Туры по району отелей (Tours Through the Hotel District)**

Ассоциация с районом в туризме: для туристов отель – это убежище от шума и суеты горящего отпуска.

Сравнение с тестированием приложения: место, где можно отвлечься от основной функциональности и популярных фич, и проверить что-то второстепенное.

**Тур, отмененный из-за дождя (The Rained-Out Tour)**

Ассоциация в туризме: иногда мы планируем длинную прогулку, но внезапно начавшийся дождь заставляет вносить коррективы – отменить прогулку, остановиться, чтобы согреться и высушить промокшую одежду.

Сравнение с тестированием приложения: отменяйте любой начавшийся процесс; отменяйте любым способом – кнопкой “Cancel”, закрыть приложение (через Alt+F4, диспетчер задач), закрыть вкладку или окно Ctrl+F4, нажать ESC, в браузере остановить загрузку или перейти на страницу другую страницу или нажать кнопки “Вперед”, “Назад”, “Обновить”, “Остановить”. Задача тестировщика:

* убедиться, что отмена в целом возможна
* что отмена действия не вызывает проблем в работе приложения
* что данные не теряются и не повреждаются (здесь лучше смотреть на приложение не только “снаружи”, через UI, но и изнутри – через структуру приложения, через БД
* что действие, которое мы отменяем, можно вызвать повторно и успешно его выполнить

Типичные баги:

* критичные и некритичные функциональные ошибки
* проблемы целостности данных
* юзабилити ошибки о невозможности отменить процесс

**Тур лежебоки (The Couch Potato Tour)**

Ассоциация в туризме: обычно в группе есть какой-нибудь турист, которому всё безразлично. Он стоит в сторонке, ему неинтересно, он пассивен, неэнергичен и думает “зачем я вообще платил за этот тур?”.

Сравнение с тестированием приложения: идите по “дефолтному” пути: не изменяйте значений, установленных по умолчанию, не заполняйте поля, если в них нет строгой необходимости, не переходите по рекламе, пытайтесь перейти между экранами, не нажимая кнопок и не вводя данных, не беспокойтесь о части ELSE в конструкции IF-THEN-ELSE, не совершайте комплексных действий – идите по пути “наименьшего сопротивления”.

Типичные баги:

* проблемы юзабилити
* функциональные ошибки

#### **Туры по неблагополучному району (Tours Through the Seedy District)**

Ассоциация с районом в туризме: опасный для туризма район, бандитский, где в любой момент может случиться что-то нехорошее – украдут, нападут, обзовут и т.п.

Сравнение с тестированием приложения: места приложения, подверженные атакам недобропорядочных пользователей.

**Тур саботажника, диверсанта (The Saboteur Tour)**

Ассоциация в туризме: приехала группа туристов к исторически важному собору, а он закрыт вдруг для посещений. Оказывается, какой-то шутник просто повесил табличку “Закрыто”.

Сравнение с тестированием приложения: задача тестировщика – попробовать подорвать (подстрекнуть) работу приложения любым возможным способом. И для этого можно выполнить 4 главных шага:

* Начните выполнять какое-то действие
* Определите какие ресурсы нужны системе, чтобы выполнить это действие
* Удалите или ограничьте системе доступ к этим ресурсам.
* Повторите действие

Примерами условий для тестов могут быть: отсутствие интернета, недостаточное количество оперативной памяти, удаленный файл, который нужно прочитать, отсутствие необходимых для выполнения операции прав, поиск или запрос несуществующего объекта, запуск и работа приложения на изначально проблемном окружении, подкладывание в систему битых файлов или данных, параллельная работа с другим приложением за конфликтующие ресурсы.

Типичные баги:

* креши
* непонятные для пользователя по формулировке сообщения об ошибках
* функциональные ошибки средней критичности

**Антисоциальный тур (The Antisocial Tour)**

Ассоциация в туризме: иногда, возникает желание или необходимость делать всё наперекор своему супругу, включить режим “капризный”. Он(а) идет с туристической группой в бар, а второй половинке не хочется. Но зато, когда все уже выходят из бара, вдруг появляется желание. Все идут на море, а он(а) в номер отеля; все в отель, а он(а) на море.

Сравнение с тестированием приложения: работать так, как нормальные пользователи делать не будут, выполнять всё, что противоречит логике приложения, вводить данные, которые должно быть запрещено вводить.

**Антисоциальный тур подразделяется на 3:**

*Противоположный тур (opposite tour).* Предполагает вводить или выбирать те значения, которые пользователь маловероятно, что выберет. Например, в плейлист добавить 10763 песен, или отправить на печать 12 копий. В этом туре речь идет именно о тестировании на позитивных, но маловероятных данных.

*Криминальный тур (crime spree tour)*. Предполагает вводить недопустимые (нелегальные) значения – это и sql инъекции, и ввод текста в цифровые поля, и подмена данных.

*Тур с неправильными поворотами (wrong turn tour).*Предполагает выполнять действия в неправильной, нестандартной для пользователя последовательности, менять действия местами. Например, сначала указать адрес доставки, а потом выбирать товар; сначала в пустом плеере включить shuffle песен, а затем запускать сами песни и т.д.

Типичные баги:

* креши и другие функциональные ошибки
* уязвимости в защищенности
* непонятные для пользователя по формулировке сообщения об ошибках
* проблемы в юзабилити

**Обсессивно-компульсивный тур или тур невротика (The Obsessive-Compulsive Tour)**

Ассоциация в туризме: вряд ли есть такая экскурсия, где водитель автобуса отчаянно ездит туда-сюда по одной улице, чтобы не пропустить чего-то важного, но в тестировании подобное поведение может окупиться.

Сравнение с тестированием приложения: повторяйте одно и то же – вводите один и тот же текст раз за разом, выполняйте многократно какое-то действие или последовательность действий. Например,

* осуществите поиск в интернет-магазине
* откройте информацию по любому товару
* положите товар в корзину
* повторите шаги 1-3 очень много раз

А можно просто осуществлять постоянно поиск. Это тоже относится к этому туру.

Типичные баги:

* некритичные функциональные ошибки
* проблемы производительности
* утечки памяти

## Лекция №6 Тестирование требований

### **Введение**

Требования – это первое, на что смотрит команда проекта, это фундамент для проектирования и разработки продукта. Допущенная в документации ошибка или неточность может проявиться в самый неподходящий момент. Очевидно, что гораздо проще устранить дефект в паре строк требований, чем позже переписать несколько сотен (или даже тысяч) строк кода.

[Тестирование требований](https://dist.belstu.by/mod/assign/view.php?id=57437) направлено на то, чтобы уже на начальных этапах проектирования системы устранить максимально возможное количество ошибок. В перспективе, это позволяет:

* значительно снизить итоговую стоимость проекта;
* улучшить качество продукта;
* сохранить нервы всей команде.

### **Уровни и виды требований**

#### **Бизнес-требования**

Выражают цель, ради которой разрабатывается продукт (зачем он нужен, какая от него ожидается польза). Результатом выявления требований на этом уровне является общее видение (vision and scope) — 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.п.

**Примеры:**

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

#### **Пользовательские требования**

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

**Примеры:**

* варианты использования (use cases);
* пользовательских историй (user stories);
* пользовательских сценариев (user scenarios).

#### **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, нюансам 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.

**Примеры:**

* В процессе инсталляции приложение должно проверять остаток свобод-ного места на целевом носителе.
* Система должна автоматически выполнять резервное копирование дан-ных 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, надёжность, 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.

**Примеры:**

* При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов.
* Ни при каких условиях общий объём используемой приложением памяти не может 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 взаимодействия разрабатываемой системы с другими системами и операционной средой.

**Примеры:**

* Обмен данными между клиентской и серверной частями приложения при осуществлении фоновых AJAX-запросов должен быть реализован в фор-мате JSON.
* Протоколирование событий должно вестись в журнале событий операци-онной системы.
* 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 базы данных и особенностей её использования.

**Примеры:**

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

### **Дефекты в требованиях**

**Выделяют 9 критериев качества требований:**

1. корректность;
2. недвусмысленность;
3. полнота;
4. непротиворечивость;
5. упорядоченность по важности и стабильности;
6. проверяемость;
7. модифицируемость;
8. трассируемость;

**Корректные требования.**

Набор требований к программному обеспечению является корректным тогда и только тогда, когда каждое требование, сформулированное в нем, представляет нечто, требуемое от создаваемой системы.

**Недвусмысленные требования.**

Требование является недвусмысленным тогда и только тогда, когда его можно однозначно интерпретировать. Хотя главным свойством любого требования по праву считается корректность, неоднозначность зачастую представляет собой более сложную проблему. Если формулировка требований может по-разному интерпретироваться разработчиками, пользователями и другими участниками проекта, вполне может оказаться, что построенная система будет полностью отличаться от того, что представлял себе пользователь.

**Полнота набора требований.**

Набор требований является полным тогда и только тогда, когда он описывает все важные требования, интересующие пользователя, в том числе требования, связанные с функциональными возможностями, производительностью, ограничениями проектирования, атрибутами или внешними интерфейсами. Полный набор требований должен также задавать требуемый ответ программы на всевозможные классы ввода — как правильные, так и неправильные — во всевозможных ситуациях. Помимо этого, он должен содержать полные ссылки и подписи всех рисунков, таблиц и диаграмм набора требований, а также определения всех терминов и единиц измерения.

**Непротиворечивость набора требований.**

Множество требований является внутренне непротиворечивым, когда ни одно его подмножество, состоящее из отдельных требований, не противоречит другим подмножествам. Конфликты могут иметь различную форму и проявляться на различных уровнях детализации; если набор требований был написан достаточно формально и поддерживается соответствующими автоматическими средствами, конфликт иногда удается обнаружить посредством механического анализа. Но, скорее всего, разработчикам вместе с другими участниками проекта придется провести проверку множества требований вручную, чтобы удалить все потенциальные конфликты.

**Упорядочение требований по их важности и стабильности.**

В высококачественном наборе требований разработчики, клиенты и другие заинтересованные лица упорядочивают отдельные требования по их важности для клиента и стабильности. Этот процесс упорядочения особенно важен для управления масштабом. Если ресурсы недостаточны, чтобы в пределах выделенного времени и бюджета реализовать все требования, очень полезно знать, какие требования являются не столь уж обязательными, а какие пользователь считает критическими.

**Проверяемые требования.**

Требование в целом является проверяемым, когда каждое из составляющих его элементарных требований является проверяемы, т.е. когда можно протестировать каждое из них и выяснить, действительно ли они выполняются.

**Модифицируемый набор требований.**

Множество требований является модифицируемым, когда его структура и стиль таковы, что любое изменение требований можно произвести просто, полно и согласованно, не нарушая существующей структуры и стиля всего множества. Для этого требуется, чтобы пакет требований имел минимальную избыточность и был хорошо организован, с соответствующим содержанием, индексом и возможностью перекрестных ссылок.

**Трассируемые требования.**

Требование в целом является трассируемым, когда ясно происхождение каждого из составляющих его элементарных требований и существует механизм, который делает возможным обращение к этому требованию при дальнейших действиях по разработке. На практике это обычно означает, что каждое требование имеет уникальный номер или идентификатор. Возможность трассировки имеет огромное значение. Разработчики могут использовать ее как для достижения лучшего понимания проекта, так и для обеспечения более высокой степени уверенности, что все требования выполняются данной реализацией.

#### **Существуют различные методы тестирования требований:**

1. **Метод просмотра** (универсальный метод, выполняется бизнесаналитиком или тестировщиком):

- Ознакомление с требованиями;

- Проверка требований по критериям качества;

- Оформление дефектов в виде комментариев/вопросов.

2. **Метод экспертизы** (выполняется при участии команды из бизнесаналитиков, представителей заказчика, разработчиков, лояльных пользователей, тестировщиков):

- Планирование;

- Обзорная встреча;

- Подготовка;

- Совещание;

- Переработка и оформление изменений в требованиях;

- Завершающий этап.

3. **Метод составления вариантов тестирования** (выполняется тестировщиком). Варианты тестирования занимают промежуточную позицию между User Case и Test Case, помимо использования для тестирования требований в дальнейшем легко расширяются до Test Cases и составляют основу тестовой документации.

### **Оформление вопросов по требованиям**

* Вопросы пишите короткими и простыми. Это повышает скорость ответа на них;
* Вопрос не должен содержать «или». Отвечая на вопрос, респондент может ответить "да" и будет непонятно, на какую именно часть вопроса он отреагировал;
* Формулируйте вопрос так, чтобы ответ на него был максимально коротким. Это повышает скорость ответа на них;
* Обдумывайте ответ, который можно получить. Иногда вопросы могут быть расплавчатыми или не отражать суть проблемы. Продумав предварительно ответ, вы сможете четче сформулировать вопрос;
* Предлагая улучшения, подкрепляйте их фактами и весомыми доводами. Используйте для этого любые доступные источники информации.

#### **Примеры**

**Требование**

Приложение должно быстро запускаться.

**«Плохие» вопросы**

* «Насколько быстро?» Рискуете получить ответы в стиле «очень быстро», «максимально быстро».
* «А если не получится быстро?» Этим вы рискуете удивить или разозлить заказчика.
* «Всегда?» «Да, всегда». Хм, а вы ожидали другого ответа?

**«Хорошие» вопросы**

* «Каково максимально допустимое время запуска приложения?»
* «Допускается ли фоновая загрузка отдельных компонентов приложения?»
* «Что является критерием того, что приложение закончило запуск?»

**Требование**

Опционально должен поддерживаться экспорт документов в формат PDF.

**«Плохие» вопросы**

* «Любых документов?» Ответы «да, любых» или «нет, только открытых» вам всё равно не помогут.
* «В PDF какой версии должен производиться экспорт?» Вопрос хороший, но он не даёт 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 пользователю 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, 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 этот набор?»