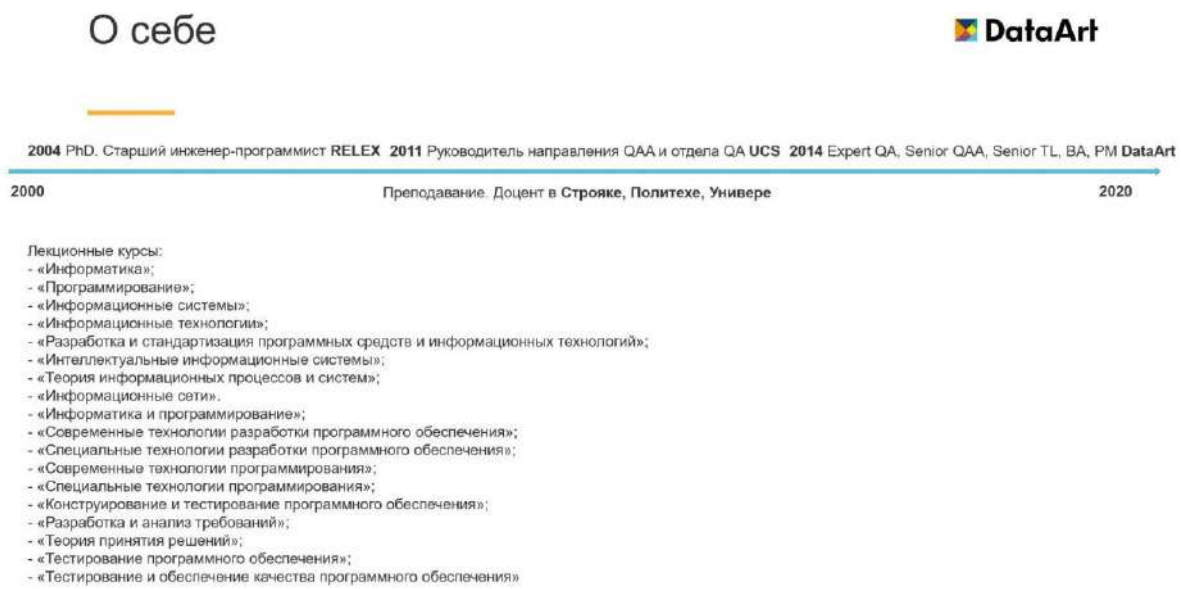


Моя карьера началась с того, что я закончил кафедру Радиофизики, устроился в Релэкс. Первый же проект сочетал в себе тестирование, программирование и техническое писательство. В Воронеже не было курсов по тестированию в 2004 году, научиться этому можно было только из Интернета, но в основном на собственном опыте. Переход в тестирование произошел благодаря тому, что мне понравилось автоматизированное тестирование.

Автоматизация про которую не будем

DataArt

В курсы лекции сказано, что мы будем рассказывать про автоматизированное тестирование, в целом это правильно, потому что курс лекций достаточно короткий, вас готовят как мануальщиков, но тем не менее и по первому слайду, и по второму хотелось сказать два вывода:

1. Посещать лекции.
2. Делать домашние задания.

Автоматизация - настоящая магия, особенно опять же в начале двухтысячных годах, когда этот этого не было фактически. Представь себе само программирование является в каком-то смысле магией, то есть мы делаем абсолютно волшебные вещи, а тут ещё и фактическим эмулируем действия пользователя. Почему вам в эту сторону иногда стоит поглядывать? Две большие причины: мануальщики зачастую из-зо дня в день выполняют одни и те же действия, например, нужно проверить сайт, разработка которого ведется в вашей команде, что он хотя бы просто откликается на наш запрос, нужно зайти, залогиниться, что-то минимальное проверить, и это делать достаточно часто. Вот хотя бы подобное скрипты научиться писать, будет полезно.

Кроме этого в те моменты, когда вы вырастите в карьере, возможно, вас повлечёт в тимлидерство в проекте. И нужно уметь представлять, что именно делают автоматизаторы вашей команде, которой вы руководите, какие примерные сроки, эстимейты их задачи. Поэтому это направление тоже нужно держать в уме, несмотря на то, что изначально вы входите в мануальное тестирование.

Информационные технологии повсюду

DataArt

Зачем вообще нужно тестирование и информационные технологии?

Уже есть умный дом, возможно в скором времени, появится такси без водителя. Становится понятно, что информационные технологии везде, и если в них находятся какие-то ошибки, они могут привести к чему-то страшному. Вот что собственно символизирует следующий слайд.

Печально известные баги

DataArt

Mariner 1

Стадион «Хартфорд Колизей»

Аппарат лучевой терапии, медицинский ускоритель Therac-25

Это три реальные ситуации, самые известные баги, которые нанесли довольно серьезный ущерб. Первая картинка - ракета, когда штаты хотели впервые сделать межпланетное путешествие, слетать на Венеру в шестидесятых годах. Произошло интересное и печальное, антенна, которая управляла данной ракетой после пуска перестала получать нормально команды из-за радиопомех, система автоматически переключила на резервное управление, то есть внутренний компьютер ракеты, а там совершенно маленькая ошибка в коде программы обеспечения. Маленькая чёрточка,

которой не хватало, привела к тому что ракета стала криница после старта и её пришлось ликвидировать, потому что побоялись, что упадёт туда, куда не нужно. Это цена порядка 20 млн долларов. И всего лишь из-за того, что данную систему тестировали не во всех режимах, то есть всегда была успешно, что справлялась одна антенна. Второй режим, который запасной, и по идее должен быть не менее тщательно проверен, но решили, что это необязательно и не проверили.

Вторая картинка - стадион сверху сложился, опять же из-за ошибки программистов. Ошибка была не в коде, а в допущениях при разработке самой системы, то есть современным языком ошибка со стороны бизнес-аналитика, который написал неправильные требования по системе. То есть средства проектирования подобных структур, стадиона, содержала допущение, что учитывался вес только крыши. В данном случае на крыше был всего лишь снег, и все опоры не выдержали, как карточный домик сложились друг за другом. Обошлось без жертв, но порядка 90 млн долларов убытка.

Третья картинка более печальна, т.к. связана с гибелью людей. Аппарат лучевой терапии, в котором было 2 режима: один настраивающий с 100 кратным превышением в рентгенах, второй рабочий. Изначально был специальный датчик который определял пациент на месте или нет, то есть можно ли давать такие большие дозы облучения. Потом его поменяли на программу, а в этой программе случилось переполнение чисел, в результате 4 человека получили смертельные ожоги, пару человек тяжёлые травмы. Поэтому профессия тестировщика очень важная и учиться этой профессии крайне полезно, чтобы подобных ситуаций не возникало.

Попробуем разобраться с начальной терминологией. Иногда путают ошибки, дефекты, бумаги, сбой, отказы. В чём же различие? По сути ошибка - это ошибка какого-то человека, говорят, что людям свойственно ошибаться, то есть программист при написании кода допустил какую-то ошибку, которая привела к дефекту в программном коде, или можем его назвать баг. И после выполнения программного кода происходит сбой или отказ системы. Это 3 разных этапа становления на одной ошибки. Я не нашел источников, но где-то мелькало, что существует команды, которые работают в принципе без тестировщиков, в частности что-то связанное с Министерством обороны США по заказам. Но по сути своей эти товарищи просто-напросто являются одновременно и

программистами, и тестировщиками, умеют смотреть на обе стороны медали, и зарплаты их такова, что в они не возмущаются, что им приходится работать чуть больше, чтобы качество было чуть лучше. И ответственность там достаточно большая, если такой человек делает ошибку, то его просто увольняют. В остальных сферах, остальных компаниях, обычно ситуация не такая жёсткая, есть распределение работ: есть программисты, есть тестировщики, которые учатся мыслить нестандартно, находить нестандартные ситуации, которые приводят к отказам и сбоям. И естественно чинить подобные ситуации на ранних стадия, до того как данная программа попадает уже в эксплуатацию.

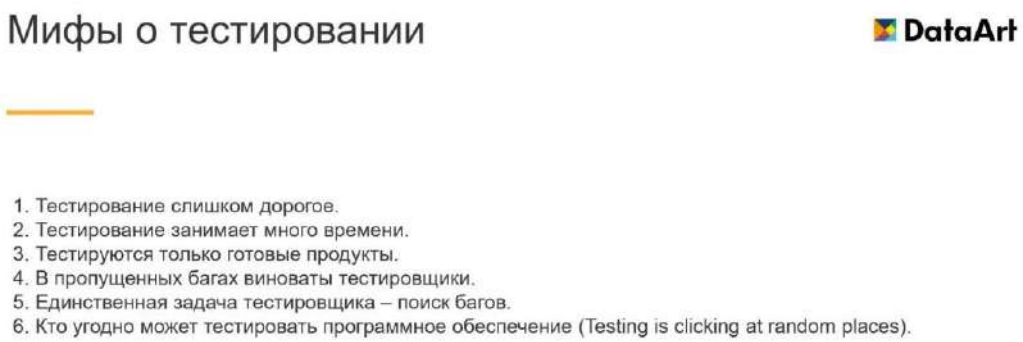
Следующий слайд показывает определения, что такое тестирование?

ISTQB - самый известный орган сертификации тестировщиков, данный сертификат признан во всём мире. Их много разного уровня: foundation level, потом ответвление в сторону тестировщиков-аналитиков, в сторону тимлидерства, автоматизации, направление интересно, почитать обязательно стоит, прикладываю ссылку на английский и русский аналог. Как вы думаете, почемц эти определение настолько разные? ISTQB, swebok или та, которая в конце она по сути перефразирование предыдущего более простыми словами. В чём разница в двух определениях? Это два разных тестирования по сути, первый - это полномасштабная, скорее ближе к Quality Assurance, которой будете проходить на следующей лекции. Второе тестирование - это та часть, которая относится непосредственно к поиску багов. Здесь также затронуто замечательное определение - верификации, это узкого определение тестирования, которое говорит о том, что это на самом проверка соответствия между ожидаемым фактическим результатом на каком-то правильно выбранном там наборе тестов по специальным методикам.

Верификация и валидация. Верификация больше смотрит в сторону требований, то есть мы должны проверить как программное обеспечение соответствует тем требованиям, которые обсуждались, заапрувлены заказчиком, подробно расписаны. Валидация - это взгляд со стороны конечного пользователя или со стороны заказчика на то, что мы сделали, произвели, действительно удобно этим пользоваться, действительно отражает, те требования пользователей к данному продукту, и действительно хочется ли данным ПО пользоваться впредь.

Верификация проверяет какой-то артефакт, то есть наше программное обеспечение или документацию на соответствие требованиям описанным и оговоренным ранее, то есть смотрит в прошлое. Заказчик, который читает и утверждает спецификацию, по которой уже было обсуждение, он по-сути проверяет и проводит верификацию. Валидация проверяет артефакт на возможность его дальнейшего использования, здесь получается взгляд в будущее. Верификация ПО это более общее понятие, чем тестирование, целью верификации является достижение гарантий того, что верифицируемый объект, требования или программный код соответствует требованиям, причём реализован без каких-то не предусмотренных заранее функций и удовлетворяет всем возможным спецификациям и стандартам. При этом он включает в себя различные инспекции тестирования кода, анализ результатов тестирования, формирование различных отчётов по проблемам, получается так, что процесс тестирования в узком смысле - это часть часть процесса верификации.

В случае верификации вопрос на который мы ищем ответ: строим ли мы продукт правильно в соответствии с теми требованиями, которые у нас есть. А в случае валидации, строим ли мы правильный продукт, который удобен для пользователя, отвечает его нужным.



Несколько мифов о тестирование.

1. Тестирование слишком дорогое

Понятно, что требует вложений, но это быстро окупается, если вспоминать те примеры, которые были на предыдущих слайдах. Поэтому вкладываться в тестирование на ранней стадии - это значит очень выгодно вкладываться, т.к. исправлять ошибки, которые сделаны на стадии требования, потом попали в архитектуру, попали в код, зачастую бывают гораздо дороже. И уж тем более если эти ошибки попадают в программное обеспечение, которые доходит до эксплуатации.

2. Тестирование занимает много времени.

Оно естественно будет занимать время, тестирование происходит параллельно всем остальным процессам. Как только бизнес-аналитики напишут конкретные требования, девелоперы берут в разработку эти требования к созданию прототипов, в это время тестировщики пишут тест-кейсы, готовится к моменту, когда данный продукт будет готов.

3. Тестируется только готовые продукты.

Если в Waterfall классическом стадии тестирования начинается после стадии девелопмента. Тестировщики не начинают тестировать ничего до того, как продукт полностью готов. То в Scrum или Kanban, в подвидах Agile всё гораздо быстрее происходит. В течение того же спринта у разработчиков, тестировщики уже получают какие-то фичи на тестирование, поэтому процесс устроен таким образом, что время на тестирование стараются минимизировать. Нет ситуации, когда простаивают девелоперы, чтобы они дождались другую команду, все процессы успешно параллелятся.

4. В пропущенных багах виноваты тестировщики.

Утверждение не совсем тоже неправильно, зачастую это так и бывает, тестировщики пропускают баги, потом получаются неприятные последствия от этого, но тем не менее, чаще всего это происходит, когда не хватает каких-либо ресурсов. Не хватает времени у тестировщиков, не хватает самих тестировщиков в команде, приходится проходить не все тестовые случаи, выбирать наиболее приоритетные. Это проблема ресурсов и зачастую сам проект диктует, чтобы в продакшн попадало не настолько качественное,

как хотелось бы тестировщикам. Иногда это оправдано, иногда нет. Всё зависит от самого проекта, если у нас программное обеспечение для тех же самых ракет или для работы с большими финансами - это одно, если у нас сайт по продаже на небольшие суммы, возможно, качества данного продукта не требуется много проверок, как в первом случае.

5. Единственная задача тестирования - поиск багов.

До того, чтобы баги найти, нужно сделать много полезной работы. Начиная от поиска багов в требованиях, потом пишутся тестовые сценарии и только потом тестируются. На этом работа тоже не заканчивается, найденные ошибки оформляются максимально подробно и качественно, чтобы их исправление занимало, как можно меньше времени. Кроме этого существуют крайне полезные репорты, которые позволяют как заказчику в любой момент времени убедиться в том, что в текущем состоянии у продукта, так и команде, куда нужно уделить больше усилий, стоит выпускать что-то следующее или сделать небольшую паузу, или откат рефакторинг кода, фиксинг, и только потом двигаться дальше, чтобы баги не накапливались.

6. Кто угодно может тестировать программное обеспечение.

Это тоже не совсем так. Тестирование сейчас фактически целая наука, которую мы мы начинаем изучать, а продолжим с вами на практике.

Кто такой тестировщик? Его требуемые качества.

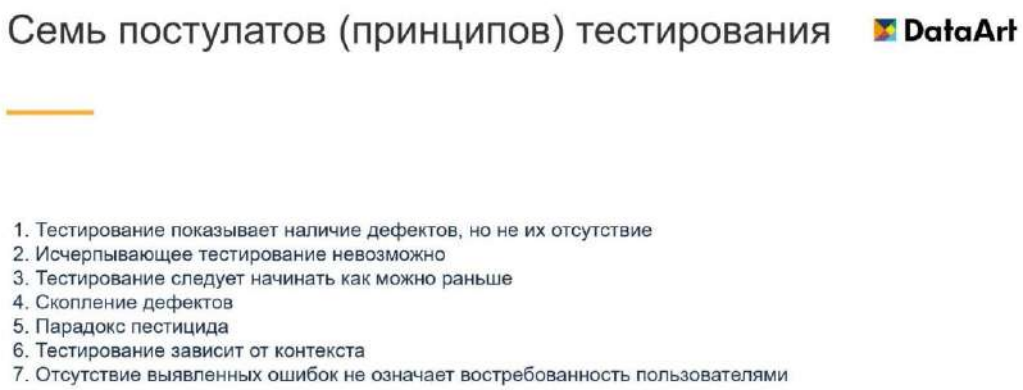
Во-первых, должно быть желание. В любой профессии должно быть желание учиться профессии, быть в ней, развиваться, иначе ничего не получится. Например, прыгнуть из тестировщика в девелопера возможно, и многие так делают через автотестирование. Поэтому должен быть интерес, определённый склад характера. С чем можно сравнивать профессию тестировщика? Детектив, который ищет убийцу, тестировщик, который ищет баг. Второе сравнение тоже уместно, с писателями и критиками. С одной стороны тестировщики не делают новые продукты, с другой стороны они обучают своими отзывами программистов делать продукты лучше. Помогают, в том числе обучают, потому что коллеги-программисты со временем начинает делать более добротный код. В некоторых командах работа Quality Control Quality Assurance заставляет действовать более аккуратно.

Интуиция бывает очень полезна. Например, фирма в ней один тестировщик делает всё по науке, пишет из кейсы, чек-листы, всё проверяет тщательно и правильно. Но иногда требуется интуиция, чтобы находить нетривиальные ошибки.

Должна быть база, то есть нужно понимать, что такой компьютер, как работает внутри, что такое компьютерные сети. Иногда на review практикантов проскальзывают вопросы, что такое утечки памяти. Ответ: не отрабатывает какой-то сборщик мусора, либо это открытые файлы были, либо переменные, объекты, которые потом не уничтожаются после своей работы. Эффект следующий, что программное обеспечение проверяется таким образом, что работает в течение длительного времени, смотрят за параметрами системы. И где есть утечки памяти, там постепенно эту память занимает.

По сетям, понимание, что такое клиент-серверное взаимодействие, тонкий и толстый клиент, чем они друг от друга отличаются, что такое API, виды запросов. База, которая в работе пригодится, если образование гуманитарное, то нужно найти пару полезных книжек, например, "Знакомство с компьютером. Устройство, выбор, конфигурация" (Леонтьев Виталий Петрович). Почитать подобную литературу, чтобы обучение проходило легко.

Базовые знания программирования, чтобы представлять, что такое условный оператор, циклы, почему мы проверяем граничные условия - один из методов написания тест-кейсов, почему там копятся ошибки.



Из базового тестирование это 7 постулатов тестирования.

1. Тестирование показывает наличие дефекта, но не отсутствует.

Находя дефекты, мы можем говорить о том, что они есть, но не сможем доказать, что нет каких-то других. Например, если за каждую итерацию находятся новые баги, то нужно продолжать тестирование. Если команда начинает находить одни и те же, и новые отсутствуют, то можно останавливать.

2. Исчерпывающие тестирование невозможно.

Оно возможно, если мы имеем, что-то простое, например, кнопку ввода и вывода. Но если мы сталкиваемся с чем-то более сложным, возникают трудности. Поэтому придумывают различные техники тестирования, иначе бы тестировалось всё полным перебором.

3. Тестирование следует начинать как можно раньше.

Цена дефекта растет в прогрессии от того, как быстро удалось ликвидировать. Одно дело пофиксить на стадии требованиях, понять что данное требование противоречиво, в двух его местах написаны противоположные вещи. Это более выгодно нежели довести до опытной эксплуатации, а там что-то сломается.

4. Скопление дефектов.

Тестирование должно быть сосредоточено пропорционально ожиданием и реальной плотности дефектов по модулю, большая часть дефектов обнаруженных при тестировании или повлекших за собой основные количество сбоев системы, содержится в большом количестве модулей. Например, бизнес-аналитики могут неидеально описать бизнес логику, и сложные модули обычно содержат больше ошибок. И к ним нужно с большим недоверием относится тестировщикам. Кроме этого существует эффект, когда одни ошибки перекрывают другие. Мы находим баг, а под ним там ещё несколько, которых мы не видели, потому что эта часть приложения не работала из-за первого бага.

5. Продукт пестицида.

Кейсы устаревают, и на одних и тех же тестовых сценариев не находятся новых багов. Их требуется постоянно обновлять, совершенствовать, добавлять новые тестовые данные, если это предусмотрено. Одно из отличий тест-кейсов и чек-листов в пользу последних, тест-кейсы настолько подробны и содержат зачастую тестовые данные, по чек-листам проверяют в более свободном стиле, но этот подход дает больше найденных багов. Чтобы этого не было тест-кейсы нужно постоянно обновлять.

Иногда перед сдачей приложения тестировщиков может не хватать, и в команду зовут сторонних тестировщиков для помощи, как раз им детально прописанная инструкция в чек-листе очень может помочь.

6. Тестирование зависит от контекста.

Разные системы требуют разного уровня качества на выходе. Поэтому соответственно с этим тратить ресурсы по проверке всего приложения, достигая идеального качества, хороший способ разорить побыстрее заказчика и свернуть проект. Другая ситуации, когда мы тестируем особо рисковые приложение, где риски - это большие финансы, либо там жизни и здоровья людей, здесь уровень качества совершенно другой, соответственно подход к тестированию более тщательный и скрупулёзный.

7. Отсутствие выявленных ошибок не означает востребованность данного приложения пользователем.

Это практически разница между верификацией и валидацией. То есть можно сделать идеальное приложение по требованиям, которые будут не нужно ни одному пользователю. Например, ваше приложений требует 10 телодвижений, чтобы пользователь получил тот самый результат, который он хочет, а конкурирующие приложение это же делает за 2 клика. Естественно, если подобные виды тестирования с привлечением пользователей не проводились, то осознать это заранее невозможно.

Цели и задачи тестирования. Что мы хотим выдать в результате наших действий?

1. Обнаружение дефектов и последующее повышение качества.

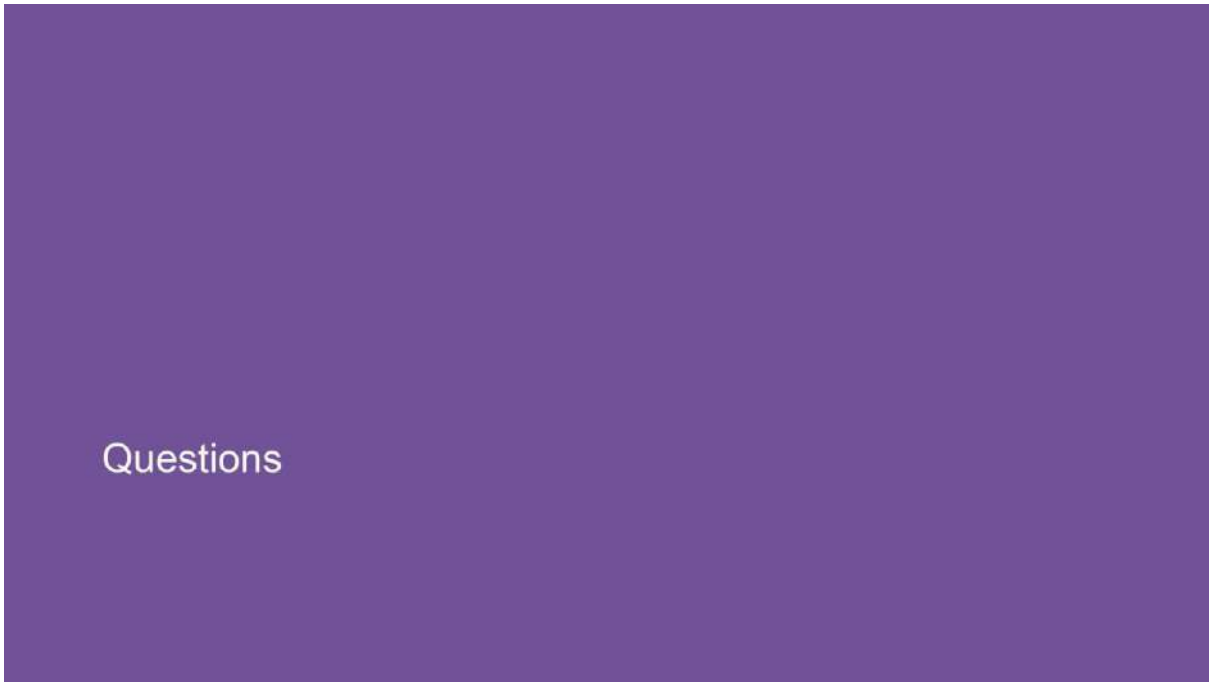
Чем больше дефектов было найдено, тем меньше попадёт в опытную эксплуатацию, и качество программного обеспечение будет лучше, соответственно будут довольны все, и заказчик, и разработчик.

2. Получение уверенности в уровне качества.

Существует определенный метрики, которые позволяют нам сказать, что данное программное обеспечение является достаточно качественным для заявленного уровня качества на данном проекте. И мы можем выпускать очередную версию этого программное обеспечение. И как раз третий пункт говорит о том, что данную информацию можно и нужно предоставлять различным третьим лицам для принятия решений, руководителям проекта, заказчику, для того чтобы понимать на какой стадии мы находимся и планировать делать бизнес-планы по поводу выпуска.

3. Предотвращение дефектов.

Относится ближе к Quality Assurance - это предотвращение дефектов, чтобы в принципе дефекты, т.е. это работа с документацией, с требованиями на непротиворечивость, неполноту тестируемости.Чтобы данные проблемы выявились на ранней стадии, до того, как они стали дефектами.



1. Как добавляют тестировщиков в сформированную команду?

Процедур много разных, то есть есть альфа-, бета-тестирование, особенно это пользуется популярной в Waterfall. Альфа-тестирование означает, что мы приглашаем сторонних людей для конкретного проекта. Например, DataArt - аутсорсинговая компания, и из нее для данного проекта берутся не те тестировщики, которые делали проект, а сторонние, чтобы они высказали своё особое мнение. Чтобы те баги, которые может быть тестировщики этой команды и видят, но глаза уже замылены тем, что они прошли этап согласования, их убедили, что это не так страшно. А сторонние тестировщики сказали, что это проблема, которую надо обязательно фиксить, например, очень неудобно в использовании. Обычно без тест-кейсов это уже валидация. Бета-тестирование, когда совершенно незнакомые люди, обычно конечные пользователи.

Если касаться не Waterfall, а Agile там тоже несколько сценариев. A/B тесты - специальное программное обеспечение позволяет сделать так, что одна часть пользователей, которые зашли на этот сайт, видят одним образом, а другие - другим. Затем сравниваются их отзывы, смотрят статистику, кто стал чаще заходить.

Acceptance testing - его проводит заказчик с конечным пользователем в команде на основание тест-кейсов. И когда всё проходит успешно подписывается сторонами документы, что контракт сделан, всё успешно.

Либо это выглядит, как Use Case Testing, его пишут чаще бизнес-аналитике, это удобнее для бизнеса, чем смотреть и апрувить тест-кейсы. Результатом всех подобных видов тестирования является то, что создаются отдельные инциденты на доработку ПО. Затем, как правило, они проходят change-реквесты. Т.е. контакт уже выполнен, а новые требования уже от пользователя, на них заводятся дополнительные задачи.

2. Коммуникация тестировщиков и девелоперов.

Со стороны девелопера, вам указывают на ситуацию, на недочёты, вы ошибаетесь. Нужно умение преподнести это правильно. Что нужно сконцентрироваться на результате, чем быстрее мы это исправим, тем всем станет лучше, умение договариваться в этом плане очень важно. Чтобы эти недостатки воспринимались наоборот с пониманием, что мы поработали баги, выявили, как можно раньше. Второй

момент не мене важен, не только с девелопером нужно договариваться, а с менеджментом. Объяснять, что несмотря на подгорание сроков, есть серьезные проблемы, с которыми нельзя будет двигаться дальше, никаких релизов не будет. В каком-то роде, тестировщик работает именно на заказчика и отчитывается перед ним, нежели на проект. Но умение договариваться со всеми заинтересованными лицами очень важно.

3. Нужна интуиция, но насколько при этом нужен пользовательский опыт?

Всегда есть возможность получить пользовательский опыт внутри DataArt, у нас очень много практик, но если заказчик “экзотический”, мы ранее с подобным не сталкивались, то он должен понимать, что нам потребуется время, чтобы войти в проблематику. Опыт обязательно нужен, девелопер может знать кусочек своего кода и делать постоянно в нем изменения, хороший тестировщик должен знать систему целиком.

4. Сообщать ли разработчику напрямую или делать баг-репорт?

Я бы предпочел создавать мелкие баг-репорты. Т.к. разработчик может забыть, если ему сказать лично. Иногда заказчик задается вопросом о том, сколько денег тратиться на тестирование, насколько это необходимо. Например, если вы работаете в Jira, там легко отследить, о скольких багах заявил девелопер, а о скольких тестировщик. И если в вашей команде именно такая система, то делать баг-репорт обязательно.

5. Различие между QA и QC?

В продуктовых компаниях есть чёткие разграничения между QA и QC. DataArt из-за своей аутсорсинговой направленности очень разный от проекта к проекту. Мы не действуем по единому шаблону или стандартам, зачастую подвержены мнению заказчика, чего он хочет от тестирования.

6. За что могут уволить?

Увольняют за непрофессионализм, за непопадание в свои собственные эстимейты, за то, что человек только делает вид, что работает. А если сделал ошибку, то он приобретет опыт и в дальнейшем не будем совершать. При наборе в проект, будут брать сотрудника с лучшей репутацией. Тогда возьмут одного, а второй посидит в idle (без проекта), можно заниматься обучением на это время. Но если долгое время не берут в проект, то уволят, но к этому нужно долго идти. С другой стороны, в idle можно провести год, но это под вас не могут найти проект, а вы имеете хорошие отзывы.

7. Вопрос про документацию и требования.

Если на проекте существует документация, которая заапрувлена, по которой девелоперы разрабатывают ПО, по которой внутри команды проводились специальные митинги. И тестировщик не высказал нареканий к этой документации, т.е. требования - на первом месте, а собственное мнение по поводу удобства использования никто не запрещает завести тестировщику багом. И это задача, несмотря на то, что она пришла не от заказчика, а от человека внутри команды, будет также обсуждаться, ей будет поставлен приоритет.

8. Что такое системы контроля версий?

Почему ее не было раньше? Раньше один программист мог сделать задачу в одного, сейчас над проектами работают большие команды, код нужно хранить, над одними исходниками работают несколько программистов. Нужно их изменения склеивать воедино и ничего не терять.

Если тестировщик умеет писать unit-тесты, то он тоже будет пользоваться системой контроля версий. Если автотестировщик, то они тоже этим пользуются. Также удобно отслеживать, какой сейчас там build, кто последний закоммитил. Чтобы отразить затем

в документации, также, что именно мы тестируем, где именно ошибки, кто может быть виноват, после какого комитта всё произошло.

9. Сравнение проектов в геймдеве и аутсорсинговой компании?

Речь о поддержке. Т.е. в геймдеве продолжается работа после выпуска постоянно, поддержка в течение всей “жизни” этой игры. В DataArt также есть саппорт, после окончания проекта оставляют небольшую команду. Это может быть на полную ставку, они продолжают с этим проектом жить, сколько нужно заказчику.
Или контракт выполнен, заказчик ушел, через полгода у него появились новые идеи по развитию - он вернулся. И скорее поддержат его уже другие люди, т.к. те, которые с ним работали окажутся уже в других проектах.
Существуют понятия, как overtime (переработки), к ним никто не может принудить, сотрудник соглашается, если хочет зарабатывать больше. Или part-time, когда работают на новом и помогают старому проекту.

10. Полезная литература.

Здесь тяжело посоветовать. Например, многие советуют “Testirovanie Dot Kom, Ili Posobie Po Zhestokomu” Романа Савина. Содержит всё необходимое, как стартовая книга вполне подходит. В дальнейшем для фундаментальных знаний подходит ISTQB.

11. Зачем в себе сочетать много технологий?

Очень интересно менять направления, так от проекта в проект не устаешь. Везде разный кайф.

12. Почему скопление багов находится в определенном модуле?

Во-первых, почему в определенных точках? Потому что на границах условий, ставят неточные условия (больше, больше или равно, меньше, меньше или равно). Во-вторых, в одном и том же модуле, потому что он сложный (например, сложная бизнес логика: скидка за то, что ты пришел в день рождения, за то, что ты постоянный клиент и пришел в определенный день недели).