БАЗОВЫЕ ЗНАНИЯ ТЕСТИРОВЩИКА ВЕБ-ПРИЛОЖЕНИЙ

Охапкин Вадим
Охапкина Марина

Марина Охапкина
Вадим Охапкин

Базовые знания тестировщика веб-приложений

http://www.litres.ru/pages/biblio_book/?art=12954782

Аннотация

В книге кратко описаны:

– суть профессии;

– подходы к тестированию отдельных элементов приложения;

– советы по написанию отчетов о найденных ошибках;

– алгоритм проверки работоспособности приложения на продуктовом сервере.

Профессия Тестировщик

Если Вы еще не определились, кем хотите стать, или ищете работу, но предложенные вакансии Вас не устраивают, то обязательно рассмотрите такую профессию, как “Тестировщик программного обеспечения”. Она позволит Вам влиться в сферу ИТ, даже не имея профильного образования. Работа в ИТ всегда оплачивается выше, чем в других отраслях, вакансии есть в любом городе, Вы не будете сокращены в кризис, и Вам не придется работать на морозе. Сейчас существует большое количество ИТ-компаний, ведущих международный бизнес, поэтому работа в них еще и престижна. Работая тестировщиком, Вы всегда будете развиваться профессионально и сможете освоить другие профессии в ИТ. Работая в международной компании, Вы еще и улучшите свои знания английского языка.

Все, что от Вас нужно – это сообразительность и любознательность. На собеседовании Вам предложат пару задачек на логику и просто пообщаются с Вами, чтобы лишь убедиться, что Вы адекватный и целеустремленный человек. Базовые знания, необходимые для быстрого старта, собраны в этой книге.

Главная задача тестировщика – проверить, что программа работает так, как этого хотел заказчик (клиент). Другими словами, он должен проверить, что работают все описанные в документации функции, при их выполнении не возникает ошибок, а также нет ничего лишнего. Обо всех несо-

ответствиях, ошибках и прочих проблемах тестировщик сообщает программисту, который будет их устранять. На профессиональном сленге любую проблему в работе программы называют багом (с английского bug – жук). Согласно Википедии, по одной из версий, в 1946 году учёные Гарвардского университета, тестировавшие вычислительную машину Mark II Aiken Relay Calculator, нашли мотылька, застрявшего между контактами электромеханического реле, и Грейс Хоппер произнесла этот термин. Извлечённое насекомое было вклеено скотчем в технический дневник с сопроводительной надписью: «First actual case of bug being found» ("первый реальный случай, когда был найден жук"). Еще одно модное словцо, крепко засевшее в лексикон ИТ-специалистов – это фича (с английского feature – особенность, свойство, фишка). Большинство команд разрабатывают программы итеративно, то есть вначале пишут минимальный базовый функционал, а затем понемногу его расширяют. Так вот, каждое такое небольшое изменение в программе и называют фичей. Теперь, пользуясь сленгом, можно сказать, что задача тестировщика – это тестировать фичи и заводить баги.

Давным-давно, когда написание программ и пользование ими было уделом лишь группы ученых в крупных научных центрах, тестированию не уделяли столько времени, сколько сейчас. Если у пользователей возникали проблемы, то они писали письмо издателям с описанием проблемы. В ответ они получали по почте дискету с исправлениями. С ростом

количества пользователей издателям становилось все труднее обрабатывать все их запросы. Им приходилось всё тщательнее проверять свои программы перед их распространением, чтобы не получить кипы гневных писем. Также с ростом количества издателей у пользователей появился выбор и они стали делать его в пользу издателей, делавших более качественное ПО. Программисты стали тратить существенную часть своего рабочего времени, занимаясь проверкой работоспособности. Для того, чтобы проверить работоспособность приложения, не обязательно знать, как оно устроено и как его написать. Поэтому, в помощь программистам стали давать людей, не имеющих опыта программирования, но имеющих опыт пользования программами. Это позволило программистам уделять больше времени написанию кода и исправлению ошибок, а также облегчило поиск новых сотрудников, так как сократился список требований к кандидату. Так появилась профессия "Тестировщик программного обеспечения". Впоследствии на плечи тестировщика легли и другие задачи, возникающие в процессе разработки ПО: сбор требований, ведение документации, поддержка пользователей. Чем больше тестировщик сможет выполнить работы, которую делал бы программист, тем больше он будет получать зарплату. Обычно зарплата тестировщика составляет около 75% от зарплаты программиста того же уровня. Поэтому постоянно расширяйте спектр ваших навыков, чтобы достичь этой цифры или даже превысить её. Большинство

книг по тестированию советуют тестировщикам никогда не говорить: “Это не моя работа”.

От программистов нередко можно услышать, что во всех бедах виноваты тестировщики. Не стоит принимать это всерьез и думать, что они хотят свалить на Вас всю вину за любую оплошность. Из этой фразы нужно извлечь только один вывод: большая ответственность – это большие полномочия. Если Вы столкнулись с серьезной проблемой, то бейте во все колокола. Трясите всех, кто может помочь Вам в ее решении, особенно тех, кто говорит, что Вы в чем-то виноваты. Навыки общения с людьми и умение работать в команде (soft skills) для тестировщика важны даже больше, чем его компетенция в тестировании. Например, тестировщики терпят фиаско в своей карьере из-за нерешительности подойти и сообщить о серьезной проблеме (возможно возникшей по их вине), из-за того, что не смогли убедить, почему она столь критична или из-за того, что побоялись выглядеть глупо, попросив помощи.

Поиск работы

Поиск вакансии нужно вести по ключевым словам ”Тестировщик”, “Специалист по качеству” или “QA”. QA (Quality Assurance) – название профессии “Тестировщик ПО” на английском языке. Откликаясь на вакансию, вышлите резюме. Ниже представлен список навыков, наиболее интересных потенциальному работодателю. Если Вы обладаете чем то из этого списка, то упомяните об этом в резюме:

Навыки программирования, в том числе написание скриптов (PowerShell или bat-файлы);

Опыт работы с базами данных (администрирование и написание запросов к базе);

Опыт работы с CMS (Content management system). CMS – это программы, позволяющие быстро создавать сайты по заранее подготовленным шаблонам (joomla, wordpress, bitrix);

Администрирование OS семейства Windows или Linux;

Знание английского языка;

Опыт написания технической документации;

Техническое образование;

Игровой опыт (Только для игровых проектов)

На самом деле, эти навыки вовсе необязательны, но их наличие будет плюсом при рассмотрении вакансии.

На собеседовании на должность тестировщика чаще всего встречается три типа вопросов:

1) Как бы Вы протестировали тот или иной предмет. Вся суть этих вопросов выяснить, как кандидат понимает суть тестирования. Почему-то распространено ошибочное мнение, что хороший тестировщик обязательно должен что-то сломать или вызвать ошибку в работе программы. Это не совсем правильно. Конечно, полезно узнать слабые места в приложении, но в первую очередь, нужно убедиться, что оно корректно выполняет все основные функции. Например, плохо, если калькулятор ломается при делении на ноль, но

куда хуже, если он выдает 5 при умножении 2 х 2. Например, Вас попросили рассказать о том, как Вы будете тестировать стул. Хороший кандидат скажет: “сначала я сяду на него и посижу 5 минут", плохой скажет: “я кину его с 3-х метровой высоты ”.

2) Что Вы будете делать, если у Вас возникла проблема. Например, у Вас есть ноутбук и компьютер. Оба подключены к роутеру. Роутер подключен к интернету. На ноутбуке есть интернет, а на компьютере нет. Ваши действия? Цель такого вопроса – выяснить не только ваши знания, но и насколько эффективно Вы решаете проблемы. Лучше всего здесь будет ответ типа: «попробую сделать то-то и то-то. Если не получится, то поищу решение в интернете. Если и 30 минут поиска не даст результата, то обращусь за помощью». Важно показать здесь Ваше стремление преодолеть проблему, попытаться сначала сделать это самостоятельно, а лишь потом просить помощи.

3) Логические задачи. Чтобы выяснить, какие действия привели к возникновению ошибки и найти кратчайший путь для ее воспроизведения, иногда приходится хорошенько поломать мозг. Поэтому на собеседованиях дают задачки, чтобы проверить, насколько сообразителен кандидат. Если решить задачку не получается, то хотя бы делайте вид, что стараетесь решить её. Давайте больше версий, делайте предположения. Не стесняйтесь просить подсказку. Вот пример типичной задачки: у Вас есть два шнура (фитиля). Каждый

шнур, подожженный с конца, полностью сгорает дотла ровно за один час, но при этом горит с неравномерной скоростью. Как при помощи этих шнуров и зажигалки отмерить время в 45 минут?

В остальном собеседование не отличается от собеседования на любую другую специальность. Желаю удачи и терпения в поиске работы.

Состав команды

Если Вы успешно прошли собеседование, то, скорее всего, Вас познакомят с наставником, который будет вводить Вас в курс дела. Обычно их называют менторами (с латинского – учитель, воспитатель). Их назначают из числа наиболее опытных тестировщиков. В небольших организациях ментор будет совмещать свою основную работу и общение с Вами. Он познакомит Вас с основами теории тестирования и даст несколько тестовых заданий. Возможно, Вам предстоит сдать внутренний экзамен. После успешной сдачи экзамена Вас уже направят на работу над проектом.

Большинство команд состоит только из программистов и тестировщиков. Ведущий программист (dev lead) – главный среди программистов. Он распределяет задания, просматривает код, написанный другими программистами (code review), оценивает трудозатраты, пишет наиболее сложные участки кода, выполняет выкладку кода на демонстрацию заказчику и на продуктовый сайт. У тестировщиков также

есть свой лидер (test lead). Его основные задачи: написание тестовой документации, повторное тестирование (overview testing), оценка трудозатрат на тестирование, редактирование багов, заведенных другими тестировщикам.

В зависимости от специфики проекта могут быть и другие роли. Например, в крупных финансовых приложениях в состав команды также входит аналитик. Его главные задачи: сбор, анализ, формализация и согласование требований к системе. Другими словами, управление требованиями на протяжении всего их жизненного цикла. Если возникают противоречия, то аналитик ищет пути решения, и своими выводами он делится с клиентом.

Верстальщики помогают программистам реализовать пользовательский интерфейс – пишут разметку страницы и ее стили. Тестировщики со знанием основ программирования (SDET – Software Development Engineer in Test) пишут автоматизированные тесты.

Цикл Разработки ПО

В этой главе речь пойдет о том, какие этапы включает в себя разработка программного обеспечения и кто принимает участие в каждом из этапов. В зависимости от способа организации труда (методологии) набор этапов и их порядок может меняться.

Сбор требований. Он осуществляется через деловую переписку или личные встречи с клиентом. Ее цель: выяснить, как клиент видит готовый продукт. Очень часто задание мо-

жет быть сформулировано лишь как некая проблема, решение которой Вы должны предложить. Лучше даже предложить все возможные варианты и дать клиенту возможность выбирать. В сборе требований принимают участие почти все члены команды: программисты, тестировщики и аналитик (если он есть). Программисты выясняют, какие технологии и библиотеки можно использовать, дают рекомендации клиенту: каким образом выбранное решение повлияет на производительность сайта и на сроки сдачи. Для тестировщика этот этап наиболее важный. Лишь досконально поняв, что должно получиться в итоге, он будет способен действительно хорошо протестировать свой продукт. Поэтому на этом этапе нужно задать как можно больше вопросов, продумать и представить в голове, что должно получиться в итоге. Проверить, нет ли противоречий и не освещенных областей. Если впоследствии обнаружатся какие-то проблемы, то их решение будет сопряженно с переделками и переписыванием большого количества кода. Уложиться в сроки будет очень сложно.

Оценка трудозатрат. Примерные сроки будут даны еще на этапе сбора требований. На этом этапе нужно дать более точную оценку. Для этого новый функционал делится на части и каждая часть оценивается на основе вашего опыта. Программисты оценивают продолжительность разработки и примерное время, необходимое на починку багов, которые обязательно будут. Тестировщики оценивают продолжитель-

ность тестирования и написания документации.

Написание спецификаций. На этом этапе вся переписка должна быть упорядочена и зафиксирована в одном месте. Обычно этим занимаются тестировщики.

Написание кода. Когда все требования собраны, программисты приступают к реализации. Тестировщики не участвуют в этом процессе, но и не сидят без дела. Они готовятся к тестированию, например, пишут списки того, что должно быть проверено (Check-list), или занимаются другим функционалом.

Написание автоматизированных тестов. Чаще всего их пишут для покрытия уже готовой функциональности (регрессионное тестирование). Тесты запускают для того, чтобы проверить, что последние изменения ничего не поломали в существующем коде. Их написанием обычно занимаются опытные тестировщики со знанием основ программирования. Также сейчас становится популярной практика написания автоматизированных тестов до того, как будет написан какой-либо код – Test Driven development.

Тестирование продукта исполнителем. Это главная область ответственности тестировщика. Тестировщики последовательно выполняют все запланированные тесты и конспектируют дефекты. После устранения дефекта тестировщик должен перетестировать ту область, где он был обнаружен. Затем тестировщики выполняют все тесты по второму разу после того, как все найденные дефекты были устранены.

Это необходимо для того, чтобы окончательно убедиться в готовности продукта к сдаче.

Тестирование клиентом. Клиент также тестирует готовый продукт, чтобы убедиться, что все сделано так, как он хотел. Также клиент может найти пропущенные баги. Обо всех своих замечаниях он сообщит в письмах. Обязанность тестировщика – ознакомиться с письмами и попытаться воспроизвести проблему в своей тестовой среде. Если это баг, и его удалось воспроизвести, то тестировщик должен завести баг в баг-трекере и передать его на починку программисту. Если клиент сообщил о несоответствии продукта его ожиданиям, то тестировщик должен выяснить от клиента все детали и дать задание программисту по устранению этого несоответствия. Также бывает, что клиент не смог разобраться, как пользоваться новыми фичами. В этом случае тестировщику необходимо лишь снабдить его детальными инструкциями.

Выкладка кода в тестовую среду. Продуктовый сайт – это сайт, которым пользуются реальные пользователи. Это конечный пункт назначения. Залив на него новый код, можно столкнуться с непредвиденными проблемами. Их устранение может привести к простою сайта. Это, непременно, вызовет большое недовольство пользователей или даже приведет к финансовым издержкам. Быстрая починка багов на продуктовом сайте может принести еще большие проблемы. Поэтому, как только клиент одобрит новый код, его следу-

ет выложить в тестовую среду (Stage environment). Он представляет собой полную копию продуктового сайта (иногда даже с базами данных). Это позволит узнать, какие проблемы могут возникнуть на продуктовом сайте, а также произвести тестирование в условиях приближенных к реальным. Задача тестировщика – проверить работоспособность сайта после выкладки.

Выкладка кода на продуктовый сайт. Выкладка кода на продуктовый сайт – это наиболее ответственный этап во всем процессе разработки. Выполняется он по такому же сценарию, что и выкладка кода на тестовый сайт. Если Вы используете один сервер, то всю процедуру нужно выполнить максимально быстро, чтобы сократить время простоя сайта. Использование нескольких серверов немного снижает риски, но увеличивает трудозатраты. При таком подходе один из серверов отключают. Вся нагрузка распределяется на оставшиеся серверы, поэтому выкладку лучше выполнять в часы с минимальной нагрузкой (выясняются опытным путем). Затем обновляют код отключенного сервера, тестируют его и включают обратно. Вся процедура повторяется до тех пор, пока не будут обновлены все серверы.

Поддержка клиентов. После заливки кода на продуктовый сайт начинается горячая пора. Выявляются все пропущенные баги, сайт может начать работать очень медленно, пользовательские данные могут быть повреждены и много чего еще. Задача тестировщика: разобраться в сути этих про-

блем, найти стабильные шаги для воспроизведения, предложить стратегию по выходу из сложившейся ситуации и профилактические меры.

Начало Работы

Попадая в команду, новый тестировщик оказывается в "колыбели". Вся наиболее сложная работа проделана более опытными коллегами. Новичку, скорее всего, доверят тестирование какой-нибудь новой фичи, по которой уже собраны все требовании и написаны чек-листы (Check-list – список того, что нужно проверить при тестировании). После новичка всё еще раз протестируют. Здесь как раз можно хорошо проявить себя, поэтому вначале мы рассмотрим подходы к тестированию с наиболее распространенных элементов приложения.

Начнем с формы с несколькими полями, в которые нужно что-то ввести. Данные будут сохранены при нажатии на кнопку. Для простоты в начале на нашей форме будет только пара текстовых полей и собственно кнопка – "Сохранить" (Submit).

Первым делом нужно проверить, что форма хоть как-то работает и готова к тестированию. Для этого выполним са-

мый позитивный кейс (case). Кейс – это тестовый случай. Если весь процесс тестирования разделять на кусочки, то кейс – это минимально возможный кусочек, как квант для энергии или фотон для света. Под самым позитивным кейсом понимают действия, которые пользователи будут выполнять чаще всего. Например, в поле “Имя” пользователи чаще всего будут вводить значения типа “Иван”, “Петр” и т.д., а затем будут нажимать кнопку [Submit]. Это и будет самый позитивный кейс. Начинать тестирование всегда нужно с самых позитивных кейсов. Это непреложная истина тестирования. Начинать тестирование поля “Имя” с ввода строки типа “?><@!#” – дурной тон. Скорее всего, Вас посчитают дилетантом или человеком недалеким. Если позитивный кейс не прошел, то возвращаем эту форму на доработку с комментарием: “Плохое альфа-тестирование. Позитивные кейсы не проходят”. Альфа-тестирование – это тестирование, которое должен делать программист перед тем, как отдать новый функционал (feature) тестировщикам. Он должен проверить позитивные кейсы. Программисты нередко ленятся это делать, считая тестирование чем-то недостойным их величества, за что могут получить от руководства, ибо потратили время тестировщика впустую. Тестировщик должен искать сложные баги, придумывая разнообразные ситуации, а не открывать форму и видеть, что ничего не работает.

К сожалению, мне приходилось видеть тестировщиков, которые начинали тестировать нерабочую форму с извра-

щенных кейсов и даже заводили баги типа “Данные не сохраняются при вводе кавычек”. Подобные баги не имеют смысла, если форма не работает ни при каких условиях. Хуже того, они вводят всех в заблуждение, программист начинает повторять действия, описанные в баг-репорте, хотя не все настройки были сделаны или программа установлена неправильно. Распутать подобные ситуации бывает непросто – будьте внимательны и профессиональны.

Итак, позитивные кейсы прошли и перед нами работающая форма – введенные данные сохраняются в базу данных. Здесь нужно уточнить, что для действительно хорошего тестирования тестировщику нужно иметь доступ к тому месту, где хранятся данные, с которыми работает программа. Чаще всего в веб-приложениях таким местом является реляционная база данных. Чтобы извлечь из нее данные, нужно написать запрос на языке, который она поддерживает. Обычно это язык SQL. Освоить простые запросы можно за несколько минут. Для начала этого вполне будет достаточно. Возможно, в вашем проекте данные хранятся как то по-другому, но это не меняет сути – необходимо получить возможность контролировать, что данные действительно сохранены. Конечно, можно поверить пользовательскому интерфейсу, где после нажатия на кнопку [Submit] нам сообщат, что данные успешно сохранены, но где гарантия, что это действительно так.

Приступим к тестированию отдельных полей нашей фор-

мы. Начинать тестирование нужно с ознакомления с требованиями к готовому продукту. Возможно, не все аспекты работы отдельных элементов были освещены в требованиях в силу различных причин. Возможно, заказчик доверяет Вам самостоятельно определить незначительные детали, давая лишь общее представление о новом функционале. Например, может быть не уточнено, какое количество символов может принимать поле “Имя”. Поэтому мы будем руководствоваться здравым смыслом в определении максимальной длины ввода, и не будем раздражать заказчика большим количеством вопросов. На данном этапе мы будем считать, что все ключевые требования собраны и понятны, а остальные аспекты интуитивно понятны. О проблемах сбора требований мы поговорим в поздних главах этой книги. Далее рассмотрим подходы к тестированию основных типов полей – текстового поля, текстового поля с автозаполнением, выпадающего списка и т.д.

Текстовое поле (Input text field) – это основной элемент, предназначенный для ввода текстовых данных. Что нужно проверить:

– *Пользователь может осуществить ввод текста в поле.* Это особенно актуально, если Ваше приложение может быть запущено с мобильного устройства без физической клавиатуры. Виртуальная клавиатура может не появиться при установке фокуса в поле.

– *Пользователь может ввести с клавиатуры все разре-*

шенные символы. Бывает так, что поле имеет защиту от ввода некоторых специальных символов. Поэтому нужно проверить, что разрешённые символы не под запретом. Проверьте, что не возникает ошибок при сохранении данных, имеющих все разрешенные символы. Особенно важно это проверить, если поле должно принимать все символы, которые можно ввести с клавиатуры. Чаще всего возникают ошибки при вводе символов, которые являются зарезервированными в таких языках как XML, JSON, HTML, SQL, так как эти языки используются для передачи и хранения данных. Использование зарезервированных символов может нарушить работу приложения, если они не были преобразованы в специальную последовательность разрешенных символов (escape or encode). При извлечении данных из базы происходит обратное преобразование (decode). Весть процесс не заметен для пользователей.

Примеры того, в какую последовательность преобразуются символы в XML:

" – "

' – '

< – <

> – >

& – &

Примеры зарезервированных символов в различных язы-

ках:

XML: ‘ “ < > &
HTML: ‘ “ < > &
*JSON: ‘ “ *
SQL: ‘

– *Поле разрешает ввести количество символов, указанное в требованиях, но также не разрешает превысить это количество.* Плохо, если поле разрешает ввести большее количество символов, чем нужно. При попытке сохранить строку с длиной, превышающей максимальную для этого поля, случится ошибка или часть строки обрежется. Если количество символов не оговорено, то проверьте, что оно достаточно, руководствуясь здравым смыслом.

– *Проверьте, как сохраняется пустое значение* (в поле ничего не введено), если оно разрешено, конечно. Оно должно сохраниться в базу как NULL (специальное значение, которое используется для заполнения пустых ячеек).

– *Проверьте, что пробелы, введенные в начале и в конце строки, обрезаются (truncate) при сохранении.* Пробел может быть введен пользователем случайно и не замечен, так как он не виден. Пробелы могут вызвать проблемы, так как они могут повлиять на сортировку по этому полю, могут повлиять на результаты поиска и т.д.

Валидация

Говоря об элементах ввода, нельзя не упомянуть про та-

кое понятие, как валидация. В нашем контексте валидация – это проверка данных перед сохранением. Мы проверяем, что данные соответствуют нашим ожиданиям. Например, мы проверяем, что в числовое поле пользователь не ввел буквы. Валидация позволяет нам избежать заведомо некорректных данных и повышает стабильность работы программы. Если пользователь пытается ввести некорректные данные, то при срабатывании валидации пользователю будет выведено сообщение о том, что он сделал не так и как это исправить. Чаще всего валидация срабатывает при нажатии на кнопку [Submit], но бывают и другие способы вызвать валидацию. Об этом чуть позже. Работа валидации также должна быть описана в требованиях. Частично она вытекает из требований к типу и формату данных, принимаемых полем.

Что может проверять валидация:

– обязательность заполнения поля;
– запрещенные символы;
– формат введенных данных;
– уникальность данных;
– истинность данных (Captcha).

Валидация на обязательность заполнения поля – наиболее частый кейс. Ее работу легко проверить. Оставьте обязательное поле пустым и нажмите кнопку [Submit]. Мы должны увидеть сообщение об ошибке, например:

Также можно сделать пару кейсов, введя в поле только пробелы. Так как пробелы обрезаются, а кроме них ничего не введено, то поле станет пустым. Поэтому проверьте, что валидация на обязательность ввода игнорирует пробелы и знаки табуляции.

Большое количество обязательных полей может вызвать раздражение у пользователей, так как требует от него больших усилий. Поэтому, делая поле обязательным, задайте себе два вопроса:

1) Можно ли отложить ввод данных в это поле. Например, пользователь регистрируется в интернет-магазине. Конечно, магазину обязательно нужно знать адрес пользователя, но на этапе регистрации лучше его не спрашивать. Возможно, пользователь не захочет ничего покупать и адрес не потребуется. Но куда более неприятно будет узнать, что пользователь, увидев дюжину обязательных полей, передумал регистрироваться и иметь дело с этим магазином.

2) Можно ли заполнить обязательное поле каким-нибудь значением по умолчанию. Например, ваш интернет-магазин

делает доставку только по Москве и Московской области. Поэтому поле “город” лучше всего не оставлять пустым, а по умолчанию вписать в него самый большой город на обслуживаемой территории – Москва.

Всегда проверяйте, что валидация отсекает все ненужные символы. Даже если список разрешенных символов не оговорен, то лучше все равно добавить валидацию на специальные символы. Сложно предугадать поведение пользователей. Иногда думаешь: “Ну кто в поле Имя будет вводить угловые скобочки или одинарные кавычки?”. К сожалению рано или поздно это все равно случится. Например, к Вам попытается зарегистрироваться пользователь, пожелавший остаться неизвестным. Вместо своего реального имени он введет свой игровой ник – “Roc!<&Roll”. Данные, введенные в наше поле, могут проделать очень длинный путь от браузера пользователя на сервер, оттуда на сервер базы данных и обратно по этой цепочке в браузер другого пользователя или в другую часть приложения. Также данные могут быть использованы в другом проекте или сторонней библиотеке, используемой в вашем проекте. Предусмотреть качественную обработку специальных символов во всей этой длинной цепи крайне сложно. Какой-нибудь компонент обязательно выдаст ошибку. Выявить слабое звено на этапе тестирования не представляется возможным, так как не все компоненты готовы и собраны воедино.

Из всего выше сказанного следует один единственный

вывод: всегда разрешайте вводить пользователям только те символы, которые действительно необходимы. Кстати, иногда валидация на специальные символы бывает слишком навязчивой: “Вы ввели запрещенный символ, используйте другой”. Поэтому иногда просто не разрешают вводить некорректные символы – пользователь нажимает на клавиатуре кнопку, но в поле ничего не появляется. При таком подходе не забудьте проверить, что ничего также не появляется при вставке данных из буфера обмена:

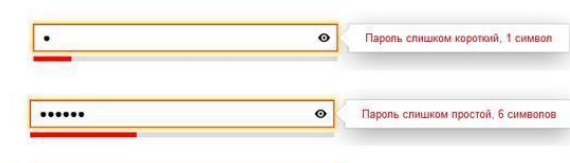
– нажимая сочетание клавиш Ctrl+V для Windows

– через контекстное меню (правая кнопка мыши в Windows)

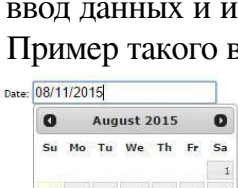
– через alt code (зажав клавишу alt, нужно набрать число от одного до 5 символов)

Некоторые поля, такие как email или пароль, могут иметь валидацию, проверяющую, соответствует ли введённая строка некоторым правилам – формату. Например, пароль должен быть длиной не менее 6 символов, иметь хотя бы одну цифру и специальный символ. Такую валидацию лучше всего проводить на лету, то есть, как только пользователь установил курсор в это поле, сразу нужно подсказать, каким требованиям должен удовлетворять пароль. Также, по мере заполнения этого поля, нужно сообщить, каким требованиям ввод пользователя уже удовлетворяет. Ниже приведен пример такой валидации. Конечно, я не берусь утверждать, что такой подход единственно верный, но как показала моя

практика, валидация на лету (on fly) очень удобна пользователям.



Для ввода даты или телефона лучше всего использовать дополнительные инструменты – виджеты, которые облегчают ввод данных и исключают ошибки в соблюдении формата. Пример такого виджета – Date Picker:



Такие поля, как User Name или Email, принимают только уникальные значения и не допускают повторений. Ведь нельзя, чтобы на сайте было зарегистрировано 2 пользователя с одинаковым именем – как их различать?

Также иногда нужно проверить, является ли пользователь человеком. Для этого с сервера пользователь получает картинку с трудночитаемым текстом – Captcha. Роботу не под силу ее прочитать, а без нее он не сможет зарегистрироваться (это как раз нам и нужно). Человек введёт текст с картин-

ки и успешно зарегистрируется. Здесь будет уместна валидация, проверяющая истинность введенных данных.

Проверить два описанных выше случая не сложно.

В первом (Email):

– вводите уникальный email и проверяете, что валидация прошла;

– вводите использованный email и получаете сообщение об ошибке.

Во втором (Captcha):

– вводите надпись с картинки и видите, что валидация прошла;

– вводите заведомо ложную надпись и получаете сообщение об ошибке.

Для проверки правильности ввода e-mail и captcha их необходимо отправить на сервер. Это нужно делать по следующим причинам:

E-mail: На вашем сайте может быть зарегистрировано очень большое количество пользователей. Отправка этого огромного списка пользователю может привести к замедлению работы браузера или к зависанию. Также есть много желающих заполучить список активных email-адресов вашего сайта и использовать их в своих корыстных целях. Злоумышленники смогут перехватить эти данные, если они будут отсылаться пользователю.

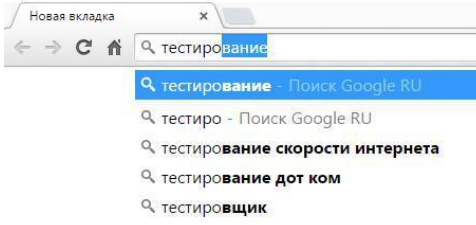
Captcha: Captcha служит защитой от автоматической регистрации. Если выслать истинное значение, то робот, вы-

полняющий регистрацию, сможет перехватить его и обойти защиту.

К сожалению, иногда всё ещё приходится видеть сайты, на которых проверка captcha и e-mail происходит только после отправки всей формы (отправляются данные со всех полей). Сервер проверит правильность данных и в случае ошибки не сохранит данные. Пользователю вернется сообщение, в каком поле он ошибся. При этом ему придется заполнять все поля еще раз, так как форма будет перезагружена. Ошибиться в вводе Captcha несложно, поэтому ситуации, когда нужно заново вводить все данные, будут случаться часто. Запомните, что в XXI веке заставлять пользователя заново вводить все данные из-за того, что он ошибся в одном месте, просто преступно. Поэтому проверки на уникальность и истинность данных всегда должны проходить на лету без необходимости перезагружать форму. Если всё же из-за каких-то ограничений на вашем сайте нет возможности избежать перезагрузки формы, то попросите программистов сохранять все введенные пользователем данные при перезагрузке страницы. Исключением могут быть только пароли (это необходимо для обеспечения безопасности).

Текстовое поле с автозаполнением (Input text with auto complete). По мере ввода в это текстовое поле пользователю будут предложены варианты, похожие на уже введённый текст. Варианты перечислены в выпадающем списке. Иногда наиболее подходящий вариант вставляют в поле и

выделяют. Выделение делают для удобства, чтобы при необходимости можно было легко удалить предложенный текст. Чаще всего такие поля делают для ввода поискового терма (Term – текст по которому выполняется поиск). Внешне они выглядят так же, как и обычные текстовые поля. Наиболее подходящий пример – это поисковая строка браузера.



Не используйте такой тип поля, если хотите ограничить ввод только заранее известными вариантами. Например, такой тип не подходит для выбора региона. Если пользователь наберет “Ив”, то ему будет предложен вариант “Ивановская область”. Даже если пользователь выберет предложенный вариант, то ничто не помешает ему продолжить ввод (даже случайно) и набрать название несуществующего региона. При сохранении такого значения произойдет ошибка в базе данных. К сожалению, ошибки в выборе типов полей встречаются очень часто, поэтому тестируйте не только сами поля, но и правильность их выбора. Тип поля, наилучшим образом подходящий для региона, будет описан далее в этой книге.

Auto complete может быть реализован браузером для всех текстовых полей, поэтому по нему не нужно заводить баги,

так как в работу браузера очень сложно вмешаться.

При тестировании текстового поля с автозаполнением нужно проверить все те же кейсы, что и для обычного текстового поля. Также нужно проверить, как работает выпадающий список. Далее представлены основные требования. Проверьте, что все они были оговорены в спецификации и соблюдены в приложении:

Источник данных. Обычно это введённые ранее значения. Для этого все значения, которые вводят пользователи в это поле, нужно сохранять в базу данных. Для проверки введите какое-нибудь значение в поле и сохраните форуму. Перезагрузите форму и начините вводить такое же значение еще раз. Убедитесь, что оно содержится в выпадающем списке.

Фильтрация данных. Предложенные варианты значений в списке должны быть отфильтрованы. Например, в выпадающем списке могут содержаться значения, у которых начало первого слова совпадает с введенным значением. Также возможны варианты когда начало любого слова совпадает с введенным значением или есть любое совпадение (вхождение) с введённым значением (например, ввели ”ова”, предложили “сл**ова**рь”)

Сортировка данных. Предложенные варианты должны быть отсортированы. Чаще всего по частоте ввода. Также встречается сортировка по алфавиту.

Количество значений. Список должен содержать опреде-

ленное количество значений. Большое количество нагромождает интерфейс. Неплохо бы иметь возможность изменять его в настройках приложения.

Выпадающий список (Drop-down menu) – предлагает пользователю выбрать одно или несколько значений из небольшого количества вариантов (обычно меньше 10). В тестировании выпадающего списка самый позитивный кейс – это проверка сохранения выбранного значения в базу. Выберите какое-нибудь значение и нажмите сохранить. Проверьте данные в базе. Также нужно проверить, что:

– все варианты, оговоренные в требованиях, представлены в выпадающем списке;

– значение по умолчанию соответствует требованиям.

Бывает так, что значение по умолчанию не оговорено в требованиях, либо Вам нужно определить его самостоятельно. Чаще всего в таких случаях выбирают первое значение из списка, либо оставляют поле пустым (заполняют его текстом “Please select…”). Во втором случае, если поле является обязательным, то при нажатии на кнопку [Submit] сработает валидация. Будьте уверены, что большинство пользователей предпочтут не трогать поле и оставят в нем значение по умолчанию. Поэтому в качестве значения по умолчанию выбирайте наиболее вероятный (популярный) вариант. Например, на сайте товаров для рыбалки в поле “Ваш Пол” должен быть выбран вариант “Мужской” так как большинство

ваших покупателей мужчины. С другой стороны, если Вы хотите, чтобы пользователи обратили внимание на это поле, то лучше оставить его пустым и не выбирать ничего по умолчанию. Делайте так как можно реже, чтобы не вызывать раздражение. Вот пример, когда это допустимо: Ваш сайт позволяет пользователям загружать фотографии. Тематику фотографии можно выбрать в поле “Раздел”. Если Вы хотите действительно разделить все фотографии на вашем сайте по разным разделам, то лучше оставить это поле пустым, чем выбрать в нем вариант по умолчанию, например – “Разное”. Иначе большинство фотографий будет загружено именно в этот раздел и смысл этого поля потеряется.

Напоследок проверьте, что список значений, доступных в выпадающем списке, хранится в базе данных (table-driven), а не зашит (hard-coded) в разметке страницы. Это актуально для списков, которые теоретически могут измениться. Такой подход облегчит добавление такого же поля с таким же набором значений в другое место вашего сайта, а также облегчит добавление/удаление новых вариантов во все поля. Что бы это проверить, добавьте еще пару значений в соответствующую таблицу. Убедитесь, что они появились в списке.

Выпадающий список с фильтром на клиенте.

Если выпадающий список имеет большое количество вариантов (как правило, больше 10), то для облегчения поиска нужного значения в обычный выпадающий список добавляют фильтр – обычное текстовое поле. Все ненужные вари-

анты отбрасываются по мере набора текста в это поле. Тем не менее, пользователь также имеет возможность раскрыть список и просмотреть его. Текст, набранный в фильтре, не может быть выбран, и пользователь может выбирать только среди предложенных вариантов. Такой тип поля наилучшим образом подходит, например, для выбора региона. Пользователь не может набрать имя несуществующего региона, не может ошибиться в наборе его имени, но может без труда найти свой регион среди нескольких десятков вариантов с помощью фильтра. Тестировать работу фильтра нужно по тому же принципу, как и выпадающий список в текстовом поле с автозаполнением.

Хорошей практикой для выпадающего списка с фильтром является автоматический выбор наиболее подходящего варианта после набора текста пользователем. Например, если пользователь набрал в фильтре “Ив”, а в нашей стране только один регион начинается с этих букв – Ивановская область, то этот вариант должен быть выбран автоматически. В этом случае пользователю не придется делать дополнитель-

ные действия для выбора региона, так как его выбор уже очевиден.

В примере с регионами страны для всех пользователей из России будет предложен один и тот же набор вариантов. Тем не менее, при каждой загрузке страницы сервер будет делать запрос к базе данных, чтобы получить этот список. Так как этот список меняется не часто, то можно сохранить его в оперативной памяти сервера – в кэше (server cache). Теперь сервер будет делать запрос в базу данных не каждый раз при загрузке страницы, а раз в несколько минут. Это ускорит загрузку страницы и уменьшит нагрузку на базу данных. Таким образом, если Вы используете выпадающие списки с большим количеством вариантов, **убедитесь, что Вы используете кеширование данных**.

Для тестирования кеширования узнайте у программиста, как Вы можете изменить период обновления (timeout) данных в кэше. Выставьте период в 5 минут и проверьте два случая:

Данные загружаются из кэша

1) Загрузите страницу;

2) Раскройте список;

3) Запомните количество элементов в списке;

4) Добавьте или отредактируйте один из вариантов в базе данных;

5) Перезагрузите страницу до истечения таймаута;

6) Проверьте, что содержимое списка не изменилось, так

как данные должны были загрузиться из кэша;

Данные в кэше обновляются

1) Перезагрузите страницу после истечения таймаута;

2) Проверьте, что изменения, сделанные в базе данных, применились к содержимому выпадающего списка.

Выпадающий список с фильтром на сервере.

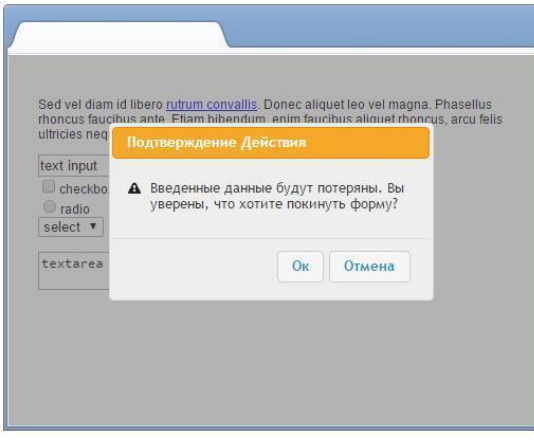
Если Вы захотите реализовать выпадающий список, в котором количество вариантов больше 200, то загрузка страницы с таким полем будет занимать значительное время. Причина – большое количество передаваемых данных. Для таких нужд лучше всего использовать выпадающий список с фильтром на сервере. При загрузке страницы такие выпадающие списки загружаются вообще без вариантов. И, следовательно, пользователи не могут раскрыть список, чтобы просмотреть полный их список. Загрузка вариантов происходит только после того, как пользователь ввел несколько символов в фильтр. При этом, чем больше вариантов для текущего поля есть в базе данных, тем больше символов нужно набрать пользователю, чтобы произошел первый запрос к серверу за вариантами. Вот почему лучше не нарушать это правило. Предположим, поле содержит 10 000 вариантов. Пользователь ввел всего лишь один символ. Браузер сделал запрос к серверу. Вполне возможно, что в ответ придет список длиной из 1000 вариантов. Такой большой объем данных

может привести к зависанию браузера. Проверьте, что при вводе различных комбинаций Вам не удалось вызвать зависание браузера. В остальном тестирование такого поля аналогично тестированию выпадающего списка с фильтром на клиенте.

Уведомление о потере данных (Prompt to Save Changes)

Хорошей практикой является появление диалога при покидании формы с введенными данными. Этот диалог должен предупреждать о потере введенных данных и требовать от пользователя подтверждения действия. Например, пользователь открывает некую форму, частично заполняет ее и случайно нажимает на какую-нибудь ссылку (кнопку) на текущей странице. Если такого диалога не будет, то страница, на которую указывает ссылка, будет загружена немедленно и все данные будут потеряны.

Обычно такой диалог имеет предупреждающий текст, и кнопки [Ок] и [Отмена]:



При нажатии кнопки [Ок] пользователь теряет данные и переходит по ссылке, при нажатии кнопки [Отмена] диалог закрывается и пользователь остается на текущей странице.

Проверку работы диалога начните с позитивного кейса: введите данные в одно из обязательных полей и нажмите на самую любую ссылку (кнопку) на странице, например, кнопка [Поиск]. Убедитесь, что диалог показан.

Теперь проверьте следующее:

вызовите появление диалога и нажмите кнопку [Отмена]. Убедитесь, что Вы остались на текущей странице без потери данных;

вызовите появление диалога и нажмите кнопку [Ок]. Убедитесь, что Вы перешли по нажатой ссылке / нажатая кнопка сработала;

убедитесь, что диалог не показан, если пользователь ни-

чего не ввел;

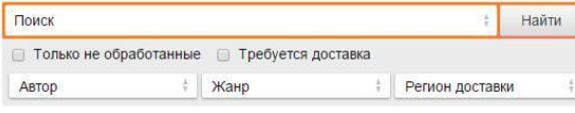
убедитесь, что диалог показан при вводе данных в каждое из полей. Проверьте каждое поле отдельно;

убедитесь, что диалог будет показан при нажатии на каждую из ссылок/кнопок на странице, которые могут вызвать потерю данных;

убедитесь, что диалог будет показан при повторном его вызове.

Уровни доступа (Access rules)

Обычно недостаточно только одной формы, позволяющей создавать записи в базе данных. Как правило, сайты имеют функционал, позволяющий просматривать список всех записей, а также фильтровать/сортировать и искать их в этом списке. Для этих целей используют поисковую панель и панель с результатами поиска (search results grid). К примеру, мы создаем интернет-магазин, а форма позволяет пользователям оформлять заказы на доставку. Вот как может выглядеть поисковая панель на таком сайте:



Поисковая панель может иметь следующие элементы:

текстовое поле для поискового терма;

поисковые фильтры;

кнопка [Search];

кнопка [Clear].

Не важно, какие элементы имеет ваша поисковая панель, в любом случае начинать тестирование нужно с простого нефильтрованного поиска. Такой поиск должен возвращать все заказы, которые есть в базе данных. Для этого все фильтры должны быть выставлены в значение “Любой”, “Все”, “All” и т.п. Поле для ввода поискового терма также должно быть оставлено пустым. После запуска такого поиска проверьте следующее:

все записи действительно вернулись из базы (Количество записей в базе соответствует количеству заказов в результатах поиска);

в результатах поиска нет удаленных и деактивированных заказов. Это может показаться странным, но во многих проектах при удалении записей из базы они не удаляются оттуда, а лишь помечаются как удаленные. На то бывают разные причины: увеличение производительности сервера баз данных или наличие возможности восстановить данные;

если на вашем сайте реализована система ролей, то нужно проверить, что каждой роли доступны только заказы, соответствующие бизнес-логике вашего сайта. Например, администратору должны быть доступны все заказы, региональным менеджерам доступны только заказы из их регионов, обычным пользователям только их собственные заказы. Проверьте не только наличие всех заказов, соответствующих текущей роли, но и отсутствие недоступных заказов;

проверьте, что при удалении и смене ролей меняется и набор доступных записей. Например, при удалении роли “Региональный Менеджер” из доступных заказов пропадают все заказы от пользователей из регионов менеджера. Также при смене региона для менеджера пропадают заказы из удаленного региона и появляются заказы из назначенного.

Поиск

Только убедившись, что действительно все записи возвращаются из базы, можно приступать к фильтрованному поиску. Тестирование фильтров нужно начинать с тестирования каждого фильтра по отдельности. Удобнее это делать, когда в базе имеется лишь небольшое количество записей. Например, наш интернет-магазин занимается продажей книг. Все книги разделены по видам литературы: “Художественная литература”, “Образовательная литература” и “Документальная проза”. В поисковую панель добавлен фильтр “Вид Литературы”. Помимо всех видов литературы этот фильтр также должен иметь вариант “Все виды”. Выставьте фильтр в одно из значений и проверьте, что все книги этого вида и доступные текущему пользователю представлены в поисковой выдаче. Убедитесь, что нет книг, не относящихся к выбранному виду. Только проверив все фильтры по отдельности, переходите к тестированию нескольких фильтров одновременно.

Перед тестированием поиска по терму уточните в требованиях, по каким текстовым полям должен осуществляет-

ся поиск. Как правило, это текстовые поля, для которых не предусмотрено фильтров. Например, запись о заказе книги имеет следующие поля:

– Номер заказа;
– Название книги;
– Вид Литературы;
– Цена;
– Заказчик;
– Дата;
– Статус.

В этом случае поисковый терм будет сравниваться с номером заказа и названием книги. Поля, по которым ведется поиск, называются индексируемыми.

Проверьте, что при поиске по терму возвращаются все заказы, где есть совпадение с термом в проиндексированных полях. Также проверьте, что не возвращаются заказы, в которых совпадения нет. Важно проверить, что индексируются все поля, заявленные в требованиях.

Обратите внимание на то, как сортируются поисковые результаты. Довольно часто клиенты забывают уточнить, какая сортировка по умолчанию им нужна. В то же время правильно подобранная сортировка облегчает поиск и пользователи реже будут ее менять. Вернемся к примеру с заказами в интернет-магазине. Рассмотрим два случая:

Менеджеры, выполняющие заказы, выполняют фильтро-

ванный поиск, чтобы проверить, нет ли новых заказов. Логично предположить, что сортировка по дате (от новых к старым) будет наиболее востребованной и может быть выбрана в качестве сортировки по умолчанию;

Пользователи ищут книгу по названию. Порой они не знают точного названия и хотят найти ее лишь по обрывку из памяти. В этом случае лучше отсортировать книги по их релевантности к поисковому терму. Релевантность – это соответствие текстовых полей поисковому терму. При сортировке по релевантности сверху списка будут результаты, наиболее точно совпадающие с поисковым запросом. Каждый поисковый движок имеет свой алгоритм подсчета релевантности. Он учитывает количество совпадений, частоту повторений, совпадение словоформы и наличие синонимов. Например, пользователь ввел в поисковое поле фразу “Уход за маленькими котятами”. Вот как могут быть отсортированны результаты поиска:

– уход за котятами;
– уход за маленькими щенками;
– котята маленькой кошечки;
– красивые котята.

Эти два случая могут быть скомбинированы. При поиске по терму будет использована сортировка по релевантности, а при поиске без терма результаты будут отсортированы по дате.

Если на вашем сайте есть сортировка числовых данных,

то нужно проверить, что числа сортируются по возрастанию или убыванию. Иногда сортировка чисел упорядочивает их по алфавиту, как будто это строковые значения, а не числа. Например, 2, 12, 20 и 100 будут отсортированы в 100, 12, 2, 20. Это, конечно же, баг.

Поисковый Движок (Search Engine)

Реализация собственного текстового поиска – непростая и трудоемкая задача. К тому же существует целый ряд поисковых движков, которые могут быть подключены почти к любому проекту. Они не только освобождают от изобретения велосипеда, но и к тому же обладают завидной производительностью (как раз этот факт чаще всего и является ключевым).

Как работает поисковый движок? Он создает собственный индекс на основании всех индексируемых данных, которые есть в базе. Под индексом будем понимать специальный файл, в котором информация хранится в упорядоченном виде. Поисковый движок просматривает свой индекс, находит совпадения и возвращает их в поисковую выдачу.

Зачастую индекс содержит не все данные. Как правило, он содержит только значения тестовых полей. Поэтому недостающая информация извлекается из базы данных при построении результатов поиска. Ниже представлена наглядная схема этого процесса.

Тестирование поискового движка можно разделить на 2 части:

Первичное создание индекса. Полное построение индекса нужно осуществлять в двух случаях. Во-первых, если Вы подключили поисковый движок к вашему проекту не с самого начала его существования и в базе уже есть данные.

Во-вторых, Вы изменили структуру данных, и поэтому данные в индексе не соответствуют действительности. Тестировщик обязан знать, как запустить процесс полного построения индекса. Обычно в рамках проекта пишется скрипт, который делает это автоматически.

После построения индекса проверьте, как работает поиск по терму. Также проверьте, действительно ли используются данные из индекса, а не из базы. Делайте это по аналогии с тестированием кэширования данных в выпадающих списках – измените данные в базе вручную и проверьте, что данные в результатах поиска не изменились. Обычно индекс не обновляется при ручном изменении данных в базе, поэтому наше тестирование будет валидным. Если же на Вашем проекте индекс обновляется даже при ручном изменении данных, то временно отключите обновление индекса. По завершении тестирования не забудьте построить индекс заново, чтобы убрать все несоответствия и включить обновление индекса, если Вы его отключали.

Обновление индекса. После каждого изменения данных в базе нужно обновлять индекс. Проанализируйте, какие есть в вашем проекте способы изменения данных. Ниже приведен список наиболее распространённых из них:

Создание

Проверьте, что новые сущности ищутся по всем индексируемым полям

Удаление

Удалённые сущности не должны находиться при поиске по терму и без терма. Такое бывает, если индекс не обновился, а удаление произошло только в базе данных.

Обновление

Сущность не ищется по старым значениям, но ищется по новым. При обновлении данных из индекса нужно удалить старые значения и добавить новые.

Копирование

Ищется не только копия, но и оригинал. На картинке снизу изображено файловое дерево. Очень часто при копировании родителя (папки) забывают проиндексировать дочерние элементы (файлы).

Клонирование (Sharing)

Клонирование – повторное использование одной и той же сущности в двух местах. Например, на вашем сайте Вы позволяете пользователям создавать документы. Вы хотите, чтобы одни и те же документы были добавлены в два места. При этом также хотите, чтобы при изменении документа он обновлялся во всех местах. Для этого делают операции кло-

нирования. Сначала создают документ в одном месте, а затем добавляют его в другое через специальное меню. Для проверки обновления индекса, как и при копировании, проверьте, что клон и оригинал ищутся. Также обновите документ в одном месте и проверьте, что индекс обновляется для него и для клона

Дополнительные кнопки

Кроме [Submit] на форме могут быть и другие кнопки (например, кнопка выбора файла). Файл загружается на сервер автоматически после его выбора. По окончанию загрузки данные в базе будут обновлены. Не забудьте проверить, что индекс также обновляется;

Перестановка (Drag and Drop)

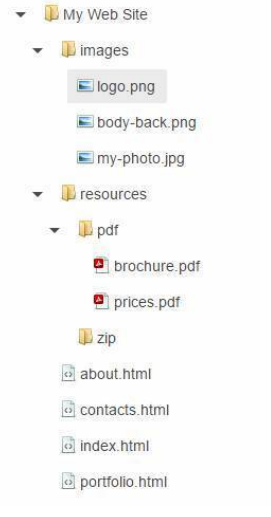
Перетаскивание и перестановка элементов в дереве (drag&drop and reorder in tree) – этот кейс актуален, если Вы индексируете данные, которые хранятся в дереве. Проверьте, что при изменении порядка и взаимного расположения элементов в дереве они не выпадают из индекса.

Версионирование (Versioning)

Предположим, что сущности на вашем сайте могут иметь несколько версий. Поиск осуществляется только среди активных версий. Проверьте, что индекс обновляется при добавлении новой версии и установке ее как основной (той, по

которой идет поиск).

Пример дерева:



Баг-Репорт

Баг-репорт – это текстовое описание ошибки и главный результат труда тестировщика. О качестве работы тестировщика судят по качеству его баг-репортов и по количеству пропущенных багов.

Вы нашли баг и хотите сообщить о нем программисту. Прежде всего, Вам нужно найти все кейсы, в которых он воспроизводится. Например, мы тестируем программу для создания и управления тест-планами. Тест-план может содержать тестовые наборы (Test Suite), а те в свою очередь содер-

жат тестовые случаи (тест-кейс, Test Case).

Допустим, Вы видите проблему с обновлением тестового случая. Вы конечно можете завести баг типа “откройте тест-кейс номер 123 и попытайтесь обновить его описание”. В этом случае программист может всего лишь исправить данные в базе, которые не позволяют выполнить обновление. Он исправит некорректные данные, но не код. Ошибка может повториться вновь. Также программист может попытаться найти источник проблемы сам, но тогда Вы отнимете его время и не заслужите должного авторитета. Поэтому постарайтесь самостоятельно выяснить шаги, которые ломают тестовый случай. Это может быть, к примеру, копирование тестового случая или ввод специфичных данных в его поля и т.д.

Найдя точные шаги для воспроизведения бага, нужно пе-

рейти к уточнению, в какой среде воспроизводится этот баг. Под средой подразумевается браузер пользователя, устройство (мобильный телефон, планшет и т.д.), операционная система и т.д. Нередко попадаются баги, которые видны только в специфичных условиях, поэтому не забывайте пробовать воспроизвести их в другой среде. Если он воспроизводится, например, не во всех браузерах, то обязательно об этом сообщите. По иронии судьбы программист будет пытаться воспроизводить его не в том браузере, в котором Вы его нашли. Обычно баг-трекеры имеют отдельное поле в описании бага для среды. Как показывает практика, если баг браузеро-зависимый, то лучше еще и в описании бага сделать на это акцент.

И так, Вы убедились, что нашли стабильные шаги для воспроизведения, и знаете, в каких условиях воспроизводится баг. Теперь можно переходить к его описанию. Для этого Вы должны придумать название, дать шаги для воспроизведения проблемы, описать текущее поведение программы и ожидаемое.

Заголовок

Заголовок должен кратко отражать суть проблемы. По хорошему заголовку можно понять, как воспроизвести баг и что должно быть в итоге. Но прежде всего заголовок должен облегчать поиск бага. Баги приходится искать очень часто, например, при заведении нового бага, Вы должны убедиться, ни завел ли уже его кто-то из ваших коллег. Например,

Вы видите такую проблему: Модератор может закрывать темы на форуме даже в тех разделах, которые ему не назначены. Все потому что ему показана кнопка [Закрыть тему]. Наверняка Вы будете пытаться найти этот баг по слову модератор или по названию кнопки, поэтому эти слова обязательно должны быть включены в заголовок.

Хороший пример:

Кнопка [Закрыть тему] должна быть доступна модератору только в темах его раздела

Плохие примеры:

Модератор может закрыть любую тему

Кнопка [Закрыть тему] должна быть скрыта

Лишние кнопки доступны модератору

Существует хорошая практика добавлять в заголовок бага название той области, где он был найден. Это также облегчает поиск и позволяет программистам скорее распределять баги между собой. Например:

Права пользователей: Закрытие темы - Кнопка [Закрыть тему] должна быть доступна модератору только в темах его раздела

После того, как фича будет отдана на тестирование клиенту, в ней делают небольшие изменения/улучшения. Иногда информацию об этих изменениях забывают внести в доку-

ментацию. Допустим, спустя некоторое время Вам потребуется тестировать эту фичу вновь. Вы откроете документ и обнаружите, что он отличается от того, что Вы видите на деле. Единственным источником актуальной информации для Вас будет баг-трекер. Просмотрев заголовки багов, Вы сможете понять, что было сделано. В этом случае Вам будет удобнее, если заголовки будут написаны в повелительном наклонении, то есть в них будет описание того, что должно быть, а не то, что видит тестировщик. Такой подход также облегчает программисту жизнь – он точно знает, чего от него хотят.

Хорошие примеры:

Ширина кнопки [Like] должна быть 50 px

Фон домашней страницы должен быть синим

Диалог "Создание нового пользователя" должен быть показан при нажатии на кнопку [Регистрация]

Плохие примеры:

Кнопка [Like] слишком узкая

Фон домашней страницы не должен быть красным

Кнопка [Регистрация] не работает

Программисты одобряют такой подход. Им становится легче понять то, чего от них хотят.

Предусловия

Иногда при заведении бага Вы можете обнаружить, что описание включает очень большое количество шагов (боль-

ше 5). Это делает баг трудным для понимания и воспроизведения. Очень часто это случается, если тестировщик пытается описать то, как он создавал какие то данные, необходимые для воспроизведения бага.

Для примера вернемся к рассмотренной ранее программе для создания и управления тест-планами. Например, мы видим баг при копировании тестового набора (test suite), в котором 10 тест-кейсов. Тестировщик может попытаться завести баг следующим образом:

1) Открой страницу создания и управления тест-планами.
2) Создай тест-план
3) Добавь в него тестовый набор
4) Добавь в тестовый набор 10 тест-кейсов
5) Выбери тестовый набор и нажми кнопку [Копировать]

В этом примере тестировщик описывает процесс создания 10 тест-кейсов. Тест-кейсы – это статичные данные, которые хранятся в базе данных. Не нужно описывать процесс создания в баге где проблема в копировании. Поэтому, если для воспроизведения бага нужны какие-то данные, то просто дайте их словесное описание. Такое описание данных и состояние системы, которое необходимо для воспроизведения бага называется предусловиями (preconditions, prerequisite). Вот как могут выглядеть предусловия для нашего примера:

Предусловия: Тестовый набор содержит 5 и более тест-

кейсов

Пример в тестовой среде: Тестовый набор ID = 241, Имя = “10ТестКейсов”

Шаги:

1) Открой страницу создания и управления тест-планами.
2) Выбери тестовый набор из предусловий
3) Нажми кнопку [Копировать]

Если баг требует предусловий, то подготовьте и пример данных из предусловий. Это упростит жизнь программисту – ему не придется делать дополнительную работу, а также на готовом примере ему будет легче разобраться в сути.

Наличие примера также подстраховывает Вас от ошибки в описании. При описании бага с более сложными предусловиями очень легко пропустить какую-нибудь важную деталь, но с помощью примера всегда можно будет разобраться в проблеме. Если баг воспроизводится на какой то определенной странице, то обязательно дайте прямую ссылку на эту страницу (иногда тестировщики описывают последовательность переходов, но это только запутывает). Не стоит заводить баги, где есть только пример, но нет описания предусловий. Пример может быть удален или кому то потребуется воспроизвести баг в другой среде, где нет этого примера.

В баг-репорт можно добавлять замечания (Notes). Обычно в них указывают дополнительные сведения, на которые нужно обратить внимание. Например, туда можно вынести

текст ошибки или выдержку из требований к той части проекта, где баг.

Также можно дополнить описание шагов для воспроизведения наблюдениями (Observation). Они упрощают понимание шагов и иногда помогают избежать повторений текста. Пример описания процесса копирования:

1) Открой страницу создания и управления тест-планами.

2) Выбери тестовый набор из предусловий

3) Нажми кнопку [Копировать]

Наблюдение 1: Появился диалог “Копирование Тестового Набора”

4) Выберите любой тест-план в выпадающем списке “Целевой тест-план”

5) Нажмите кнопку [Ок] в диалоге

Наблюдение 2: Процесс копирования начался, на странице появился спиннер

Результаты

Финальной частью баг-репорта является описание увиденного результата (Actual Results) и ожидаемого (Expected Results). В увиденном результате последовательно опишите, что произошло после выполнения всех шагов. В ожидаемом результате – то, что хотите увидеть. Ожидаемый результат должен основываться на требованиях к проекту. Например:

Увиденный результат:

1) Копия тестового набора не появилась в целевом тест-

плане

2) Спиннер не исчезает

3) Ошибка в консоли браузера

Ожидаемый результат:

1) Копия тестового набора появилась в целевом тест-плане

2) Спиннер исчезает по окончании процесса копирования

Очень часто увиденный результат и ожидаемый отличаются не только отрицанием, как в этом примере (появилось – не появилось, выполнилось – не выполнилось). Рассмотрим другой пример. Допустим, нас не устраивает набор тест-планов, доступных в выпадающем списке “Целевой тест-план”:

Увиденный результат: Выпадающий список содержит все тест-планы

Ожидаемый результат: Выпадающий список содержит только тест-планы, которые доступны текущему пользователю.

В этом случае также важно снабдить программиста примером данных, чтобы он мог проверить свой фикс, например:

Ожидаемый результат: Пользователю “tester@email.com” доступны только тест-планы “Test-Plan1” и “Test-Plan2”.

Несколько общих советов по написанию баг-репорта

– Избегайте сложных предложений и причастных/деепричастных оборотов. Разбивайте предложения на части.

– Сокращайте очевидные действия. Например, Вы тестируете графический редактор. Он доступен на странице "Редактирование и Просмотр". В баг-репорте можно опустить очевидные шаги:

Авторизуйтесь на сайте
Перейдите на страницу "Редактирование и Просмотр"
Запустите редактор

– Сводите все действия к элементарным вещам: нажатиям кнопок, перемещению мыши, вводу с клавиатуры. Например, не стоит писать фразы типа "Заблокируйте контент", лучше написать "Нажмите кнопку [Блокировать]".

– Используйте общепринятые термины. Если в вашей команде заполнитель для текстового поля все называют Placeholder, то и Вам стоит его так называть. Это особенно актуально для команд, работающих на английском языке. Со временем Вы заметите, что команда использует ограниченный набор слов. С осторожностью вводите в общий лексикон новое слово. Это может привести к двусмысленности.

– Проверьте необходимость каждого шага. Каждый лишний шаг сбивает с толку.

– Отвечайте за каждое слово. Если пишете, что баг вос-

производится для администратора, то проверьте, что он воспроизводится только для него и не воспроизводится для других ролей. Так же и с браузером – если Safari Mac, значит только Safari Mac и вы проверили во всех остальных браузерах.

– Любой баг воспроизводите с несколькими примерам. Например, пробуйте то же самое действие несколькими пользователями. Если баг воспроизводится не всеми пользователями, то попытайтесь установить разницу. Если ее нет, то, скорее всего перед Вами Data issue – это баг, который связан с нарушением целостности данных, а не с кодом. В любом случае, нужно указать аккаунт, для которого баг воспроизводится. Во-первых, программист не потратит время, напрасно пытаясь его воспроизвести. Во-вторых, Вы точнее локализуете баг, что упростит его починку.

– Заводите баг только в том случае, если поведение системы противоречит требованиям. Если в требованиях этого нет, то обсудите его в команде или напишите письмо клиенту. Избегайте субъективных суждений. Бывает так, что программисты чинят то, что на самом деле не было сломано. Потом все переделывают, по просьбе тестировщика, который думает по-другому и т.д.

– Добавляйте скриншоты к описанию бага. На них выделяйте цветной рамкой то, что Вы считаете неправильным. Другому человеку будет совсем неочевидно понять, что не так на картинке без комментария. Также добавляйте скрин-

шот того, чего Вы хотите увидеть (его можно нарисовать в графическом редакторе или подправить верстку сайта в средствах разработчика вашего браузера). В идеале совместить эти рисунки в одном.

Словарь

При заведении бага иногда не хватает слов для точного описания проблемы. Далее приведен небольшой словарь, состоящий из наиболее часто употребляемых и общепринятых терминов.

Spinner, Loading wheel – небольшая анимированная картинка, говорящая что идет загрузка, сохранение данных, поиск или любая другая операция

Progress Bar – тоже самое что и Spinner, но еще показывает на сколько процентов завершена загрузка.

Footer – Нижняя часть страницы. Обычно содержит контакты, ссылки на группы в социальных сетях, копирайт,

Title – Заголовок страницы. Определяет текст, который будет написан на табе браузера, если в ней открыта текущая страница.

Label – Подпись элемента, чаще всего элемента управления, например, текстового поля.

Tooltip – Всплывающая подсказка, появляется при наведении на элемент

Placeholder – Текст заполнитель. Он показан, когда в текстовое поле ничего не введено. Пропадает при установке фо-

куса в поле.

Credentials – Совокупность логина и пароля.

Scroll bar – Полоса прокрутки

Pagination – Разбиение одной страницы на несколько маленьких.

Popup – Всплывающее окно. Обычно является модальным, то есть невозможно как либо повлиять на страницу, если оно открыто.

Alert – Модальное окно с сообщением. Посетитель не сможет продолжить работу, пока не нажмет на кнопку "ОК" в модальном окне.

Dialog box – Всплывающее окно с вопросом и несколькими вариантами ответа. Выбор влияет на дальнейшую работу страницы.

Message – Информационное сообщение. Обычно не требует никаких действий от пользователя

Pin button – Кнопка, которая фиксирует движущиеся части страницы в одном положении.

Developer tools

Почти все браузеры имеют встроенные средства разработки. Для их включения нужно нажать клавишу F12. Большинство из них позволяет делать следующее:

Просматривать и редактировать разметку страницы (html) и стили, которые к ней применились (css)

Просматривать скрипты (javascript), которые были загружены вместе со страницей

Просматривать ошибки, которые произошли во время загрузки и работы со страницей

Просматривать всю сетевую активность, которую выполнил браузер для загрузки этой страницы, а также для ее работы

Просматривать файлы Cookie и Cache

Чем полезны средства разработчика для тестировщика и что он может проверить с их помощью:

Как уже было сказано, в средствах разработчика можно посмотреть, какие ошибки произошли при загрузке и выполнении скриптов на странице. Большинство багов сопровождаются ошибками в консоли (вкладка средств разработчика, где показаны ошибки). Существует хорошая практика указывать текст ошибки в баг-репорте, это ускоряет понимание сути проблемы и иногда освобождает программиста от необходимости самому воспроизводить баг. Сам факт наличия ошибок в консоли может быть поводом к заведению бага, даже если она не сопровождается видимыми симптомами. Как минимум, факт ошибки должен подтолкнуть Вас к более тщательному тестированию той области, где она случается.

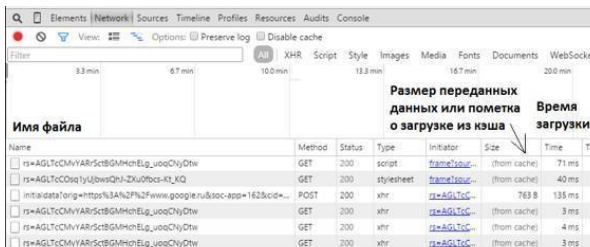
На вкладке network (сетевая активность) можно проследить, какие файлы (чаще всего картинки и скрипты) были загружены вместе со страницей, их размер и продолжительность загрузки. Программист имеет возможность настроить сайт таким образом, чтобы файлы сохранялись в память браузера (кэш) и не загружались с сервера при повторной за-

грузке страницы. Это ускоряет загрузку страницы. Если на вашей странице много картинок и скриптов, то обязательно проверьте, что такая настройка была выполнена

При работе со страницей убедитесь, что никакие скрипты не загружаются повторно.

Средства разработчика позволяют очищать файлы Cookie и Cache только для текущего сайта (домена). Это очень удобно, так как очистка файлов Cookie приводит к потере сессии. Если почистить все cookie, которые есть в браузере (Ctrl + Shift + Delete), то Вам придется повторно авторизоваться на всех сайтах.

Некоторые сайты для ускорения загрузки файлов используют CDN (content distributed network) – распределенное хранилище файлов. Пользователи загружают файлы из хранилища максимально приближенного к ним, за счет этого и происходит ускорение. При загрузке файлов убедитесь, что они загружены с CDN (Обычно CDN имеет другой адрес, нежели текущий сайт). Также на CDN загружают не только пользовательские файлы (картинки, аудио и видео) но и большие, редко изменяющиеся скрипты.



Smoke test

После завершения работы над новой функциональностью, она попадает на тестовый и далее на продуктовый (production) сайт. Процесс заливки новой функциональности в другую среду (environment) не обходится без участия тестировщиков. Что нужно проверить после заливки:

Открыть основные страницы приложения, выполнить поиск/создание объектов на них

Проверить, что не возникли проблемы, исправленные при прошлой заливке

Выполнить Smoke test. Smoke test – это быстрая проверка основной функциональности. Как правило, выполняется после заливки нового кода. Начинать его нужно с той части сайта, которая доступна конечным пользователям. Например, Вы тестируете интернет-магазин, который состоит из 2 частей. Первая часть доступна для покупателей. Она состоит из каталога товаров, корзины и страницы заказов. Вторая часть доступна для администраторов и менеджеров, которые обра-

батывают заказы. В этом случае тестирование нужно начинать с 1 первой части, так как она доступна большему количеству пользователей и ее простой обходится большими издержками для бизнеса. В рамках smoke test интернет-магазина нужно проверить, что работает просмотр товаров во всех категориях, работает добавление товаров в корзину и просмотр содержимого корзины. Также нужно выполнить заказ. В рамках Smoke test не нужно уделять внимание деталям и проверять редкие кейсы. Его цель как можно быстрее убедиться в работе основного функционала и обозначить наиболее критичные баги.

Проверить, не перетерлись ли фиксы, выполненные непосредственно на этом сайте. Бывает так, что в условиях реальной работы сайта под большой нагрузкой всплывают новые баги, пропущенные на этапе тестирования. Не всегда в их можно починить прямо на продуктовом сайте и поэтому применяют временные решения проблемы. Затем делают правильный фикс в тестовой среде, проверяют его и планируют заливку на продуктовый сайт. К сожалению, про некоторые такие костыли забывают и перетирают их при заливке. Поэтому всегда записывайте все вмешательства в работу продуктового сайта и контролируйте, что ничего не перетерлось после заливок.

Проверить, что попали все фиксы, которые были сделаны в тестовой среде, но не были сделаны на продуктовом сайте. Не обязательно проверять их детально, как Вы это делали в

тестовой среде. Достаточно убедится что код, относящийся к ним, залит.

Проверить новую функциональность. Только убедившись, что старый функционал не сломан, можно переходить к проверке новой функциональности. Так как все уже было детально протестировано в тестовой среде, достаточно выполнить только позитивные кейсы. Если в рамках новой фичи была изменена структура данных, например, было добавлено новое поле в форму заказа, то обязательно проверьте что это поле заполнено у существующих заказов. Обычно пишется скрипт который назначает всем существующим заказам новое значение. Обычно это значение по умолчанию (default value) или NULL. Уделите этому очень большое внимание, если были сделаны более сложные преобразования, например, для хранения информации о заказе была добавлена новая таблица. Если скрипт миграции данных содержит ошибки, то может произойти потеря данных. Последствия будут еще более печальными, если это обнаружится не сразу. Советую перед заливкой таких фич просмотреть и записать, как выглядят несколько наиболее характерных для вашего сайта объектов. Например, вы меняете структуру хранения корзины с товарами. Перед заливкой добавьте несколько товаров в корзину. После заливки проверьте, что все товары на месте и не появилось ни чего лишнего. Также проверьте, что вы можете оформить заказ этих товаров. Не забывайте делать бэкапы баз данных перед заливкой.

Проверить, что не ухудшилась производительность. Для этого также перед заливкой и после нее замерьте основные временные показатели: время загрузки страниц, время поиска, время переключения между поисковыми результатами.

UAT и Support

UAT (User acceptance testing) – обычно это тестирование на стороне клиента. Он проверяет, то ли Вы сделали, чего он просил и насколько качественно. Что требуется от тестировщика, если со стороны клиента поступили замечания (как правило, это баги):

Проверьте, получается ли у Вас воспроизвести баг. Манера изложения у клиента может отличаться от того, что Вы привыкли видеть в своей команде. Если не получается, то Вы можете попросить у него больше деталей (браузер, User ID, запись видео). Имейте ввиду, что работа над клиентскими багами имеет высокий приоритет. Поэтому следует отложить работу над другими фичами и заняться исследованием UAT багов.

Если Вы смогли воспроизвести баг, но, по вашему мнению, такое поведение системы не противоречит требованиям, то отправьте клиенту письмо со ссылкой на то место спецификации, которое подтверждает вашу правоту. Возможно, Вы не правильно его поняли, либо клиент сам забыл, чего просил. Это бывает, если он дал лишь общие сведения, а все детали Вы выясняли у него в письмах. Приступайте к починке бага, только получив от клиента подтверждение о

необходимости изменений.

Если Вы смогли воспроизвести баг и согласовали необходимость изменений, то оцените время, необходимое для починки (согласуйте это время с программистами) и тестирования. Сообщите клиенту о начале работ и о сроках их завершения.

Проверьте, что Вы выяснили все предусловия и способны воспроизвести баг в вашей тестовой среде. Проверьте, нет ли лишних шагов для воспроизведения, а также все ли кейсы, в которых воспроизводится баг, найдены.

Проверьте, что баг не воспроизводится после его починки и доложите клиенту о завершении работ.

Support – это поддержка пользователей (решение их проблем). Большинство проблем возникает после заливки нового кода на продуктовый сайт. Работа над багами, которые сообщили пользователи, выполняется по тем же правилам, что и над клиентскими. Разница лишь в том, что воспроизвести их бывает очень сложно и еще сложнее понять, что конкретно приводит к ошибке. Бывает, что не хватает и целого дня, чтобы разобраться, в чем дело. На это есть 2 причины:

1) От конечных пользователей к тестировщику поступает катастрофически мало информации. Часто это не последовательность действий, а лишь описание проблемы, с которой столкнулся пользователь.

2) Примеры, которые приводят пользователи, могут быть очень нагруженными и содержать множество деталей, не

имеющих отношения к делу. Например, делаете графический редактор. Пользователь столкнулся с проблемой сохранения данных. У него есть некий рисунок, в котором 100 объектов. Он добавляет 101 первый и жмет кнопку [Сохранить]. После перезагрузки рисунка мы видим, что 101 первый объект пропал, то есть данные в базе не обновились. Возможно, что Вам даже не сообщат какого типа был 101 объект. Может оказаться, что дело не в типе объекта и не в их количестве. Может даже оказаться, что проблема воспроизводится только с этим рисунком, и не воспроизводится с его копией. Еще хуже, если баг воспроизводится не стабильно.

К сожалению, нет четкой инструкции, как действовать в этой ситуации. Вот лишь несколько рекомендаций:

Проверьте, нет ли ошибок при воспроизведении бага. Загляните в консоль браузера, на сервер (обычно сервера имеют файлы – logs – куда сохраняется вся информация о серверных ошибках) или в средства слежения за ошибками операционной системы, в которой работает сервер. Текст ошибок может значительно сузить область, в которой возникла проблема.

Попробуйте воспроизвести баг в тестовой среде. Для этого обычно необходимо скопировать пользовательские данные из базы продуктового сайта в базу тестового сервера. Воспроизведение ошибки в тестовой среде позволит программисту подключить к серверу средства отладки, которые также позволят Вам и ему продвинутся к сути проблемы.

Если у Вас есть четкие шаги для воспроизведения проблемы, но Вы не знаете, что конкретно к ней привело, то попытайтесь ее детерминировать, постепенно убирая детали. В примере с графическим редактором сделайте копию рисунка. Если Вы сделали копию средствами вашего сайта и баг не воспроизводится с копией, то сравните, чем отличаются данные копии в базе от оригинала. Если воспроизводится с копией, то постепенно убирайте из нее объекты и проверяйте, не пропал ли баг. Проанализируйте, удаление какого объекта привело к исчезновению бага. Проверьте вашу гипотезу на другом рисунке.

Если баг не стабильный, то пробуйте воспроизводить его в разное время и с разной скоростью. Также оцените периодичность его возникновения. Я сталкивался с ошибками, которые воспроизводились, когда пользователь слишком быстро нажимал на кнопки. Он не дожидался ответа от сервера, это как раз и приводило к потере данных. Также некоторые баги воспроизводятся только в часы наибольшей нагрузки на сервер – часы с наибольшим количеством пользователей. Сейчас существует множество программ, которые могут симулировать большую нагрузку. Если Вы заметили, что ваш баг воспроизводится с вероятностью, например, 1/16 и чем больше Вы тестируете, тем больше Вы в этом уверены, то возможно причина генерация случайных чисел. Мне приходилось видеть баги, в которых из-за неправильно выбранного типа данных переменной в GUID отбрасывался ноль в

начале. Без этого нуля валидация на правильность формата GUID падала. Это приводило к серверной ошибке с печальными последствиями. Вероятность выпадения 0 в начале GUID – 1/16.

Деловая переписка

Как ни странно, но переписка занимает существенную часть рабочего времени тестировщика. Основные задачи для переписки:

Выяснение требований к новой функциональности и оценка трудозатрат

UAT (Тестирование на стороне клиента) – Урегулирование разночтений и починка пропущенных багов

Поддержка пользователей и исследование багов на продуктовом сайте

Электронная почта до сих пор является основным средством переписки. Рассмотрим основные правила на ее примере.

Любое письмо начинается с приветствия, например, Hello Tom. Приветствие является важной частью письма. Так как большинство писем отправляются не одному человеку, а группе, то приветствуя адресата по имени, Вы выделяете его из группы и назначаете ответственным за выполнение задания, описанного в письме. Если письмо не требует ответа или каких либо усилий, (например, Вы хотите проинформировать ваших коллег о предстоящем отпуске), то можно обратится ко всем сразу, например, Hello Team.

Большинство почтовых клиентов имеют несколько полей для адресатов:

To – Кому

CC (Carbon Copy) – Копия

BCC (Blind Carbon Copy) – Скрытая копия

Как правило, в "To" указывают тех адресатов, к которым Вы обращаетесь в тексте письма. В "Cc" указывают остальную часть команды, которая должна быть в курсе вашей переписки, скрытую копию "Bcc" чаще всего отправляют руководству или в архив, чтобы можно было восстановить переписку, например, после увольнения сотрудника и удаления его аккаунта.

Вслед за приветствием следует краткое описание содержимого письма, например:

Не могли бы Вы взглянуть на наши вопросы по новой фиче – "Название фичи"

Зачастую большинство писем отправляются руководителю команды, который распределяет задания по команде, на некоторые отвечает сам. Чтобы не перечитывать письмо полностью для того, что бы понять, о чем оно, или кто за него будет отвечать и нужно краткое описание или введение.

Во всей массе писем можно выделить две большие группы – выяснение требований и описание проблемы. При выяснении требований нужно задать вопросы, которые возникают при прочтении задания, которое прислал заказчик. Количество вопросов зависит от того, насколько детально оно про-

работано. Очень редко задание поступает в виде детально проработанной спецификации, скорее это будет пара фраз типа “Хочу добавить на наш сайт возможность посчитать статистику заказов, чтобы выявлять популярные книги и заказывать их у оптовиков заранее”. Иногда клиенты даже не знают, чего конкретно они хотят, у них есть проблема, которую нужно решить. Проработка новой функциональности – это немалая работа и она занимает существенное время. Клиенты нередко делегируют ее исполнителю, так как они не всегда являются людьми из IT и не представляют, какие еще сведения от них нужны. Также, возможно, у них есть другие задачи, и они хотят, что бы Вы все продумали, предложили дизайн, а они только подтвердили что это то, что им нужно. Исходя из этого и нужно строить общение с клиентом – Вы задаете вопрос и сразу предлагаете возможные решения.

Вопросы нужно задавать от общего к частному, постепенно уточняя детали. Для примера со статистикой нужно узнать:

На основании каких заказов считать статистику?

За какой период?

На какой странице ее можно посмотреть?

Кому она доступна?

Для каждого варианта ответа обязательно предложите опции, среди которых клиент будет выбирать, либо предложит что то свое. Не лишним будет сделать комментарии для каждой опции, которые помогут сделать выбор, например:

Опция1 – Требует меньше всего трудозатрат,

Опция2 – Обеспечивает наивысшую производительность

Если Вы не очень четко понимаете, чего хочет клиент, делаете что-то впервые, либо клиент хочет сразу очень большой функционал, то можно предложить ему разбить разработку на несколько итераций. Это поможет сконцентрироваться на главном (договорится об основных функциях, пользовательском интерфейсе) и вовремя скорректировать работу, если Вы стали делать что-то не так как он этого хотел. Например, клиент хочет, что бы Вы сделали для него векторный графический редактор, в котором можно рисовать несколько типов линий (ломаная, Безье, свободная рука) и геометрические примитивы (круг, полигон, фигуры из шаблонов). Тут Вы как раз можете предложить сделать в первом релизе по одному типу линий и примитивов. В первом релизе Вы сделаете основу, покажете клиенту, чтобы он проверил, такой ли он себе представлял ход работы в редакторе. Во втором релизе добавите дополнительные функции и расширите набор типов линий и фигур.

Если ваш вопрос подразумевает ответ “Да” или “Нет”, то не лишним будет задать вопросы типа: Должны ли мы поддерживать мобильные девайсы, если да, то какие? Это позволит сократить количество итераций при переписке. Также если Вы не уверены, что правильно его поняли, либо есть разночтения у членов вашей команды, то обязательно нужно уточнить правильно ли Вы поняли то или иное предло-

жение. Для этого опишите своими словами то, как Вы его поняли и приведите пример. Например, клиент говорит, что хочет, чтобы пользователям была показана реклама при просмотре видео ролика в тот момент, когда *начало окончания конца начала ролика началось*. Этот пример я взял из одного фильма, но он наилучшим образом демонстрирует, насколько сложно иногда бывает разобраться в требованиях. По поводу этой фразы нужно обязательно уточнить у клиента: Правильно ли мы поняли, что рекламное сообщение должно всплывать, когда видео ролик просмотрен на 3/8? Например, если продолжительность фильма 16 минут, то реклама всплывет на 6 минуте (16 * 3 / 8)?

Вторая категория писем – письма с описанием проблемы. Они пишутся в следующей последовательности:

Краткое описание функциональности, где Вы видите проблему

Суть проблемы и ее причины

Возможные пути решения или профилактики

Выводы и призыв к действию

Будет не лишним вставить описанные выше разделы прямо в текст письма и выделить их жирным. Это облегчит чтение, и позволит человеку, который в курсе дела, пропустить уже известную информацию и быстрее перейти к делу. Вот пример такого письма:

Добрый день, Федор,

Вы просили разобраться, почему некоторые пользовате-

ли испытывают проблемы при просмотре рекламных сюжетов на странице "Promotion". Вот наши результаты:

Описание страницы

На странице "Promotion" пользователям доступен список продвигаемых товаров с их техническим описанием и рекламным видео роликом.

Причины проблемы

Как мы выяснили, все пользователи, заметившие проблемы с проигрыванием видео, пользуются продукцией фирмы Apple. Предположительно пользователи пытались запустить следующее видео, не остановив предыдущее. Как Вы возможно знаете, продукция фирмы Apple не поддерживает одновременное воспроизведение двух видео или аудио на одной странице. Вот ссылка:

https://developer.apple.com/library/safari/documentation/AudioVideo/Conceptual/Using_HTML5_Audio_Video/Device-SpecificConsiderations/Device-SpecificConsiderations.html

"Currently, all devices running iOS are limited to playback of a single audio or video stream at any time. Playing more than one video—side by side, partly overlapping, or completely overlaid—is not currently supported on iOS devices. Playing multiple simultaneous audio streams is also not supported. You can change the audio or video source dynamically, however. See Replacing a Media Source Sequentially for details."

Возможные пути решения

Опция1: *Мы добавим новый скрипт, который будет вы-*

полняться при нажатии на кнопку [Play] в видео. Этот скрипт будет останавливать все видео на текущей странице и только после этого запустит текущее

Опция2: *Для пользователей продукции Apple мы будем выводить продвигаемые товары по одному, с возможностью переключатся между ними при помощи кнопок [Next] и [Back]*

Реализация каждой из этих опций потребует 2 дня на реализацию, включая тестирование. Какую из них Вы предпочитаете?

С уважением,
Карпов Сергей

В завершении письма ставится сигнатура – подпись. Она должна содержать имя и фамилию отправителя, его контакты. Очень часто приходится слышать вопрос: А зачем нужен email в сигнатуре, ведь почтовый клиент всегда показывает имя отправителя? На практике очень часто письма объединяют в треды (группа писем объединённая в цепочку), пересылают их друг другу (reply и forward), присылают письма как файл, прикрепленный к другому письму (attachment) и т.д. Если Вы получите несколько писем объединённых в тред, то при отсутствии email в сигнатуре связаться с кем-то из участников будет невозможно. Кроме email указывают и другие контакты – телефон, id мессенджеров, почтовый адрес, но они скорее для экстренных случаев, чем для

нормального режима работы. Для вашего же удобства можно указать еще ваш часовой пояс, чтобы ваши заграничные или очень удаленные партнеры реже беспокоили Вас в нерабочее время (если речь идет о телефоне) или меньше переживали, если Вы долго не отвечаете на важные письма.

Отвечая на письма, всегда объединяйте их в цепочку – тред. Вашему собеседнику будет легче понять, на какое письмо Вы отвечаете, он без труда сможет найти и почитать прошлые письма по текущей проблеме, а также он сможет оформить ответ в виде комментариев прямо в тексте вашего письма – это очень удобно и облегчает чтение. Также, отвечая на письмо, отправляйте его всем тем людям, которым оно было отправлено, когда попало к вам. Отправляя письмо, не забудьте придумать короткое, но емкое название.

Послесловие

Если Вам понравилась эта книга, и Вы хотите чтобы вышло продолжение, то можете сделать пожертвование:

Яндекс Деньги: 410012676033116

Web Money: R411931155582

По всем вопросам и предложениям обращайтесь по адресу:

Vadiminter9@gmail.com