**Анализ требований**

**Параметры тестирования документации**

**1. Четкость и ясность**

Начать тестирование требований можно с поверхностного осмотра документации. Это сложно назвать именно тестированием, но нередко уже на данном этапе выявляется немало недочетов. Начнем с обычного сценария. Вы начали читать требования и уже с первых строк у Вас возникает масса вопросов к автору (например, «Каков ожидаемый результат после нажатия на эту кнопку?» или «Что будет, если я отменю заказ?»). Это плохо. После прочтения документации не должно быть вопросов. Совсем. Требования – это как свод законов для продукта, а законы не допускают двусмысленность, «воду» и неточности. Документация должна давать предельно ясную информацию о том, как должен работать каждый отдельный модуль и весь продукт в целом. К сожалению, после прочтения большинства требований остается целый ряд вопросов.

Пример. В требованиях было записано: «В поле «Имя пользователя» могут быть введены буквы и цифры». Разработчик начал выяснять у аналитика, какие именно буквы (кириллица, латиница или арабские) и какие цифры (целые, дробные, римские) имеются в виду. После уточнения требований разработчик реализовал функционал согласно комментариям аналитика. Задача перешла в тестирование. Тестировщик не понимал, по каким критериям проверять данное поле, и тоже начал расспрашивать аналитика.

Последствия:

* Затраченное время нескольких членов команды.
* Несовпадение итогового и изначально планируемого функционалов.

Как тестировать:

* Если у Вас после прочтения требований остались вопросы – необходима доработка.
* Если разработчики часто уточняют детали в чатах – это плохой знак.

Дальнейшее («более глубокое») исследование требует гораздо больших временных затрат.

**2. Актуальность**  
Необходимость поддержания актуальности требований кажется очевидной. Однако, на некоторых проектах требования не обновляются месяцами, а то и годами. Это может быть связано с тем, что в штате нет аналитика, а у исполняющего его обязанности сотрудника просто не хватает времени. Случается и другое: требования обновляют только при наличии действительно значимых изменений, при этом различные «мелочи», в виде изменения кнопок или текстов, игнорируются.

Пример. Было решено изменить положение кнопок на странице авторизации. Аналитик не стал править документацию, а написал разработчику личное сообщение с просьбой поправить расположение кнопок. Разработчик внес правки и закрыл задачу. Во время очередного регрессионного тестирования тестировщик решил, что это дефект, и завел на него баг. Другой разработчик вернул кнопки на прежние позиции согласно документации.

Последствия:

* Время нескольких членов команды потрачено впустую.
* Итоговая позиция кнопок не соответствует ожидаемому результату.

Как тестировать:

* При наличии подобных сообщений в командном чате нужно убедиться, что обновленные требования задокументированы.
* Необходимо сравнить даты обновления технического задания и пояснительной записки с датой последнего обновления требований.

**3. Логика**  
Как следует из названия, работа системы должна быть логичной. Пользователь не может изменить настройки своего профиля или написать письмо до того, как пройдет авторизацию в системе. Звучит, опять же, элементарно, но в проектах со множеством клиентов или со сложной логикой подобные ошибки часто допускаются.

Пример. В мобильном приложении появилась необходимость реализовать функционал электронной подписи документа. Пользователю предлагалось ввести свои данные, после чего они автоматически подставлялись в шаблон документа. Приложение открывало документ и предлагало его подписать. Если пользователь понимал, что в документе есть ошибки, то исправить он их уже не мог: у него была возможность только подписать этот документ. Закрытие приложения или его переустановка не помогали – при входе пользователя в аккаунт сразу отображался тот же документ на подпись.

Последствия:

* Пользователь в бешенстве.
* Дальнейшая работа с аккаунтом без обращения в техподдержку невозможна.

Как тестировать:

* Нарисовать примерную блок-схему работы системы в соответствии с требованиями и убедиться, что в ней нет логических пробелов.
* Убедиться, что в требованиях описан необходимый основной функционал.
* Убедиться, что взаимодействие между модулями системы изложено корректно.

**4. Возможные сценарии**  
В документации должны быть подробно описаны как очевидные, так и неочевидные варианты использования системы. К очевидным (позитивным) вариантам, например, можно отнести ввод корректной пары логин/пароль. К неочевидным (негативным) – ввод некорректной пары логин/ пароль или отсутствие этих данных вовсе.

Пример. Часто из виду упускаются такие моменты, как тексты ошибок, поведение системы при потере связи, а также обработка ошибок, связанных со сторонними сервисами (например, с оплатой).

Последствия:

* При потере связи система ведет себя некорректно (отсутствие ошибок, зависание).
* Сообщения об ошибках не очевидны.
* В худшем случае возможны репутационные или финансовые потери.

Как тестировать:

* Нарисовать блок-схему отдельного модуля системы, в рамках которой обозначить все возможные условия и действия пользователя.
* Убедиться, что в требованиях есть описание каждого возможного случая.

**5. Интеграция**  
Имеет смысл выделить интеграцию со сторонними сервисами, так как здесь приходится выходить за рамки проверки документации. Перед началом разработки аналитики, как правило, изучают работу сторонней системы, а затем описывают схему взаимодействия этой системы с разрабатываемым продуктом. В данном случае, вероятность ошибки очень велика, так как ошибиться могут как аналитики, так и представители стороннего сервиса, которые консультировали или писали документацию.

Пример. На проекте необходимо было реализовать возможность авторизации через сторонний сервис. Аналитик по ошибке изучил устаревшую документацию стороннего сервиса и описал заведомо нерабочую схему взаимодействия. Разработчики начали работу, в соответствии с готовой схемой, но постоянно получали ошибки. Они «допрашивали» аналитика, а тот в спешке звонил в техподдержку стороннего сервиса и выяснял причины ошибок.

Последствия:

* Задержка разработки функционала на неделю.

Как тестировать:

* Необходимо вручную проверить, что сторонний сервис обрабатывает все необходимые запросы, в соответствии с описанной схемой.
* Проверить, указал ли аналитик корректно и в полном объеме всю необходимую для разработки информацию.