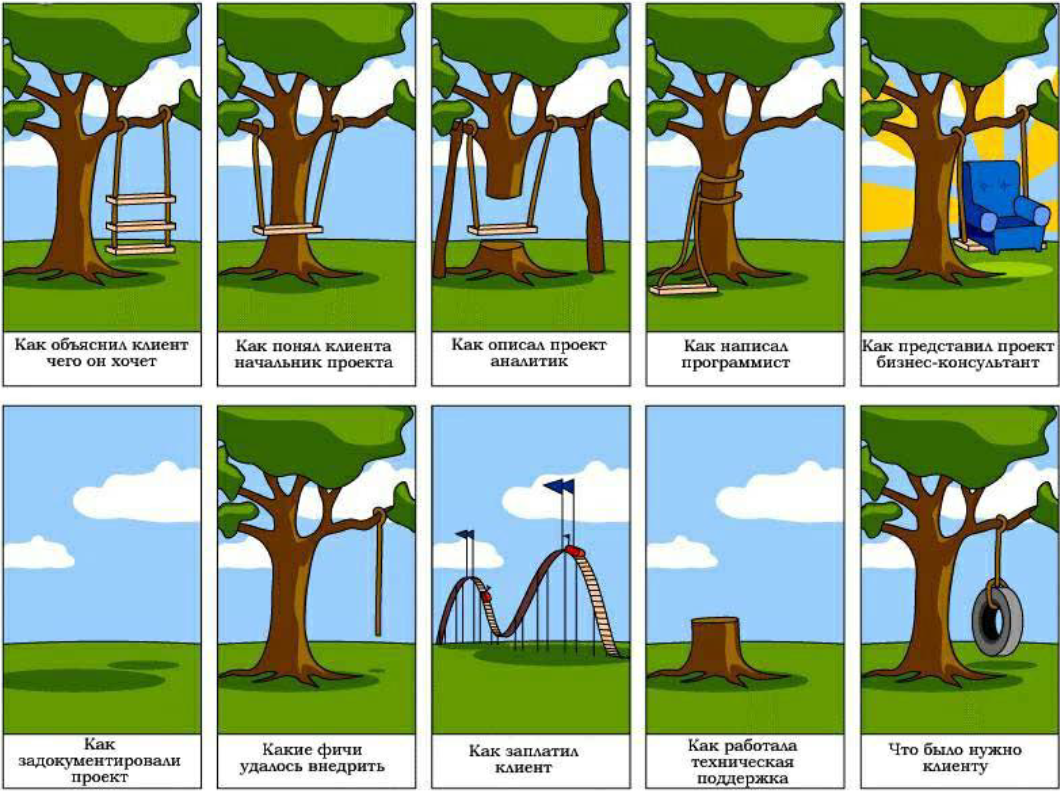
Лекция 4.Тестирование требований

# Важность требований

**Требование** – это описание того, какие функции и с соблюдением каких условий должно выполнять приложение. Требования должны быть избавлены от привязки к способам реализации, внутренней архитектуре продукта и т.п.

Если в процессе работы с проектом документации не уделено должного внимания, ситуация может стать похожей на представленную на этом рисунке.



Следует помнить о том, что стоимость исправления проблемы тем меньше, чем на более ранних стадиях работы с проектом она обнаружена. Поскольку работа с требованиями является одной из начальных стадий, именно здесь можно заранее устранить множество проблем до того, как они возникли.

Такой подход полностью соответствует принципу "Проблемы следует предупреждать, а не исправлять".

Степень важности и глубина тестирования того или иного вида документации и даже отдельного документа определяется большим количеством факторов, но неизменным остаётся общий принцип: всё, что мы создаём в процессе разработки проекта (даже рисунки маркером на доске, даже письма, даже переписку в скайпе), можно считать документацией и так или иначе подвергать тестированию (например, вычитывание письма перед отправкой — это тоже своего рода тестирование документации).

# Уровни требований

## Бизнес-требования

Бизнес –требования выражают цель, ради которой разрабатывается продукт (зачем вообще он нужен, какая от него ожидается польза, как заказчик с его помощью будет получать прибыль). Результатом выявления требований на этом уровне является общее видение — документ, который, как правило, представлен простым текстом и таблицами. Здесь нет детализации поведения системы и других технических характеристик.

Пример бизнес-требований:

* Нужен *инструмент, в реальном времени отображающий наиболее выгодный курс покупки и продажи валюты.*
* Необходимо *в два-три раза повысить количество заявок, обрабатываемых одним оператором за смену.*

## Пользовательские требования

Пользовательские требования описывают задачи, которые пользователь может выполнять с помощью программного продукта. Поскольку здесь уже появляется описание поведения системы, требования этого уровня могут быть использованы для оценки объёма работ, стоимости проекта, времени разработки и т.д. Пользовательские требования оформляются в виде вариантов использования, пользовательских историй, пользовательских сценариев

Пример пользовательских требований:

* При *первом входе пользователя в систему должно отображаться лицензионное соглашение.*
* Администратор *должен иметь возможность просматривать список всех пользователей, работающих в данный момент в системе.*

## Функциональные требования

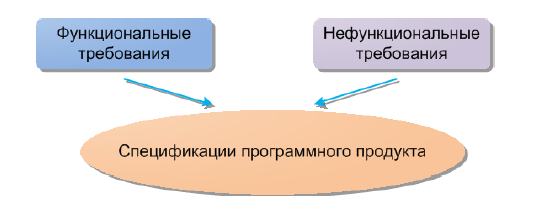
Функциональные требования описывают поведение системы, т.е. её действия (вычисления, преобразования, проверки, обработку и т.д.). Стоит помнить, что к поведению системы относится не только то, что система должна делать, но и то, что она **не** должна делать (например: «приложение **не** должно выгружать из оперативной памяти фоновые документы в течение 30 минут с момента выполнения с ними последней операции»).

Пример функциональных требований: *В процессе инсталляции приложение должно проверять остаток свободного места на целевом носителе.*

## Нефункциональные требования

Нефункциональные требования описывают специфические свойства системы (удобство использования, безопасность, надёжность, расширяемость и т.д.), которыми она должна обладать.

Пример нефункциональных требований: *При одновременной непрерывной работе с системой 1000 пользователей, минимальное время между возникновением сбоев должно быть более или равно 100 часов.*



Функциональны требования определяют, «что система должна делать».

Нефункциональные требования определяют, «с соблюдением каких условий» должно происходить это действие.

Спецификациятребованийобъединяет в себе описание всех требований уровня продукта и может представлять собой весьма объёмный документ (сотни и тысячи страниц).

# Свойства требований и параметры тестирования документации

**Требования должны быть:**

**Завершёнными**

Все важные аспекты должны быть включены. Ничто не должно быть оставлено «для будущего определения».

**Непротиворечивыми**

Требование не должно содержать противоречий как внутри себя, так и с другими требованиями

**Корректным**

Требование должно чётко указывать на то, что должно выполнять приложение. Недопустимо при написании требования предполагать, что что-то окажется очевидным. Каждый человек понимает это «очевидное» по-своему.

**Недвусмысленными**

Требование не должно допускать разночтений.

**Проверяемыми**

Требование должно быть сформулировано так, чтобы существовали способы однозначной проверки – выполнено требование или нет.

## Параметры тестирования:

##### 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 – значит, необходима доработка;
  + если разработчики часто уточняют детали в чатах – это плохой знак.

##### 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 ошибок 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.

##### 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 с 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.

##### 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.

##### 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.

# Техники тестирования требований



Одной из наиболее активно используемых техник анализа требований является взаимный перепросмотр. Суть взаимного перепросмотра требований проста: после того, как один человек создал требование, другой человек это требование проверяет.

Обычно, выделяют три уровня перепросмотра:

* Неформальный перепросмотр. Двое коллег просто обмениваются листиками (файликами) и правят найденные ошибки, которые потом обсуждаются за чашкой чая или в любое другое относительно свободное время.
* Технический перепросмотр. Это немного более формализованный процесс, требующий подготовки, выделенного времени, участия некоторой группы специалистов (желательно, из различных областей).
* Формальная инспекция. Проводится редко и в случае очень больших проектов и крайней необходимости. Описывается специальными стандартами, требует соблюдения широкого спектра правил и протоколирования результатов.

**Универсальные техники:**

* Самый простой и не требующий большого опыта способ – задавать как можно больше вопросов. Получая разнообразные ответы от разных участников проекта (как наших коллег, так и представителей заказчика), мы расширяем, углубляем и уточняем своё представление о том, что и как должно работать.
* Второй способ – создавать тест-кейсы. Когда вы видите требование, спросите себя: «Как я буду его тестировать? Какие тесты очевидно покажут, что это требование реализовано в ПС правильно?» Если с придумыванием таких тестов вы испытываете сложность – это тревожный сигнал: скорее всего, в требовании есть проблемы.
* Третий способ – рисовать. Чтобы увидеть общую картину требований целиком, очень удобно использовать рисунки, схемы, диаграммы и т.д.

**ИТОГО**

Требования – это основа разработки, на тестирование которой мало кто обращает внимание. При этом проверка документации – верный способ сохранить команде нервы и время, а проекту – бюджет. При тестировании требований важно помнить, что все члены команды должны понимать их абсолютно одинаково; это убережет от лишних правок уже разработанного функционала в дальнейшем. Кроме того, сократится количество споров внутри коллектива из-за того, что разработчик сделал одно, а аналитик имел в виду совсем другое. Конечно же, гораздо лучше, когда документацию проверяют сразу несколько человек (тестировщиков и разработчиков). Все они смогут задать правильные вопросы, исходя из своих профессиональных особенностей. Такой подход значительно повысит шансы на то, что тестирование требований будет проведено должным образом.