**Тестовые артефакты и документация (Test Deliverables/TestWare/test artifacts)**

**Виды тестовой документации?**

Внешняя документация:

Замечание – короткая записка, комментарий о небольшой неточности в реализации продукта.

Дефект-репорт – описание выявленного случая несоответствия производимого продукта требованиям, к нему выдвигаемым – ошибки или ее проявления. Он обязательно должен содержать следующие элементы:

* Идею Test case, вызвавшего ошибку.
* Описание исходного состояния системы для выполнения кейса.
* Шаги, необходимые для того, чтобы выявить ошибку или ее проявление.
* Ожидаемый результат, т. е. то, что должно было произойти в соответствии с требованиями.
* Фактический результат, т. е. то, что произошло на самом деле.
* Входные данные, которые использовались во время воспроизведения кейса.
* Прочую информацию, без которой повторить кейс не получится.
* Критичность и/или приоритет.
* Экранный снимок (скрин).
* Версию, сборку, ресурс и другие данные об окружении.

Запрос на изменение (улучшение) – описание неявных/некритичных косвенных требований, которые не были учтены при планировании/реализации продукта, но несоблюдение, которых может вызвать неприятие у конечного потребителя. И пути/рекомендации по модификации продукта для соответствия им.

Сводный отчет об испытаниях (Test summary report) представляет собой документ высокого уровня, в котором обобщаются проведенные испытания и результаты испытаний.

Политика тестирования (Test policy) – Общая цель организации при выполнении тестовых заданий описана в документе «Политика тестирования». Он создается Lead’ами и Senior’ами в группе управления тестированием совместно с руководителями групп заинтересованных сторон. Иногда тестовая политика является частью более широкой политики качества, принятой организацией. В таких случаях политика качества разъяснит общую цель менеджмента в отношении качества. Политика тестирования — это краткий документ, обобщенный на высоком уровне, который содержит следующее:

* преимущества тестирования, ценность для бизнеса, предоставляемую организации, которая оправдывает стоимость качества
* цели тестирования, такие как укрепление доверия, выявление дефектов и снижение рисков для качества
* методы измерения эффективности и результативности тестирования при выполнении целей теста
* способы для организации улучшить свои процессы тестирования

Стратегия тестирования (Test strategy) — это подход к проведению тестирования (STLC). Это относительно небольшой статический документ, который предшествует плану тестирования. Прежде чем писать объемный и подробный план, стоит формализовать некоторые базовые подходы к тестированию и убедиться, что все заинтересованные лица понимают одинаково, что и как будет тестироваться. Вероятность пропустить какую-либо тестовую активность очень мала, если существует правильная стратегия тестирования. Это самый важный документ для любой команды QA. Написание эффективного Стратегического документа - это навык, который тестер развивает с опытом. Стратегия включает:

* Роли и обязанности в команде тестирования
* Область тестирования
* Тестовые инструменты
* Тестовая среда
* График тестирования
* Сопутствующие риски

Внутренняя документация:

* Тест-план (Test plan, план тестирования) – документ, описывающий весь объем работ по тестированию, начиная с описания объекта, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения. Ответственность за создание документа плана тестирования лежит на Test Lead или менеджере по тестированию. Содержание:
  + Что надо тестировать? — описание объекта тестирования: системы, приложения, оборудования
  + Что будете тестировать? — список функций и описание тестируемой системы и её компонент в отдельности
  + Как будете тестировать? — стратегия тестирования, а именно: [виды тестирования](http://www.protesting.ru/testing/testtypes.html) и их применение по отношению к объекту тестирования
  + Когда будете тестировать? — последовательность проведения работ: подготовка (Test Preparation), тестирование (Testing), анализ результатов (Test Result Analisys) в разрезе запланированных фаз разработки
  + Критерии начала тестирования:
    - готовность тестовой платформы (тестового стенда)
    - законченность разработки требуемого функционала
    - наличие всей необходимой документации
  + Критерии окончания тестирования:
    - результаты тестирования удовлетворяют критериям качества продукта:
    - [требования к количеству открытых багов](http://www.protesting.ru/testing/bugpriority.html#openbugsreq) выполнены
    - выдержка определенного периода без изменения исходного кода приложения Code Freeze (CF)
    - выдержка определенного периода без открытия новых багов Zero Bug Bounce (ZBB)
  + Хорошие дополнения:
    - Окружение тестируемой системы (описание программно-аппаратных средств)
    - Необходимое для тестирования оборудование и программные средства (тестовый стенд и его конфигурация, программы для автоматизированного тестирования и т. д.)
    - Риски и пути их разрешения
* Тестовый сценарий (Test scenario) – последовательность действий над продуктом, которые связаны единым ограниченным бизнес-процессом использования, и сообразных им проверок корректности поведения продукта в ходе этих действий. Фактически при успешном прохождении всего тестового сценария мы можем сделать заключение о том, что продукт может выполнять ту или иную возложенную на него функцию.
  + Создание тестовых сценариев обеспечивает полное покрытие тестами
  + Сценарии тестирования могут быть одобрены различными заинтересованными сторонами, такими как бизнес-аналитик, разработчики, клиенты, для обеспечения тщательного тестирования ПО. Это гарантирует, что ПО работает для наиболее распространенных юзкейсов (use cases – сценарии использования).
  + Они служат быстрым инструментом для определения трудозатрат на тестирование и, соответственно, создают предложение для клиента или организуют рабочую силу.
  + Они помогают определить наиболее важные end-to-end транзакции или реальное использование ПО.
  + Для изучения end-to-end функционирования программы, тестовый сценарий имеет решающее значение.
* Тестовый набор/комплект (Test Suite) — Некоторый набор формализованных Test case, объединенных между собой по общему логическому признаку.
* Test case (тест-кейс) – Это артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функции или её части. Более строго — формализованное описание одной показательной проверки на соответствие требованиям прямым или косвенным. Обязательно должен содержать следующую информацию:
  + Идея проверки.
  + Описание проверяемого требования или проверяемой части требования.
  + Используемое для проверки тестовое окружение.
  + Исходное состояние продукта перед началом проверки.
  + Шаги для приведения продукта в состояние, подлежащее проверке.
  + Входные данные для использования при воспроизведении шагов.
  + Ожидаемый результат.
  + Прочую информацию, необходимую для проведения проверки.
* Чек-лист (лист проверок) – перечень формализованных Test case в упрощенном виде удобном для проведения проверок. Test case в чек-листе не должны быть зависимыми друг от друга. Обязательно должен содержать в себе информацию о: идеях проверок, наборах входных данных, ожидаемых результатах, булевую отметку о прохождении/непрохождении тестового случая, булевую отметку о совпадении/несовпадении фактического и ожидаемого результата по каждой проверке. Может также содержать шаги для проведения проверки, данные об особенностях окружения и прочую информацию необходимую для проведения проверок. Цель – обеспечить стабильность покрытия требований проверками необходимыми и достаточными для заключения о соответствии им продукта. Особенностью является то, что чек-листы компонуются теми Test case, которые показательны для определенного требования.
* Матрица покрытия требований (RTM — Requirements Traceability Matrix) - это документ, который связывает требования с тест-кейсами.
* Тестовые данные (Test Data) — это данные, которые нужны для выполнения Test case.

**Какие отличия у тест-кейса высокого и низкого уровня?**

Высокоуровневый тест-кейс (high level test case или logical test case) — тест-кейс без конкретных входных данных и ожидаемых результатов. Как правило, ограничивается общими идеями и операциями, схож по своей сути с подробно описанным пунктом чек-листа. Достаточно часто встречается в интеграционном тестировании и системном тестировании, а также на уровне дымового тестирования. Может служить отправной точкой для проведения исследовательского тестирования или для создания низкоуровневых тест-кейсов. Низкоуровневый тест-кейс (low level test case) — тест-кейс с конкретными входными данными и ожидаемыми результатами. Представляет собой «полностью готовый к выполнению» тест-кейс и вообще является наиболее классическим видом тест-кейсов. Начинающих тестировщиков чаще всего учат писать именно такие тесты, т.к. прописать все данные подробно — намного проще, чем понять, какой информацией можно пренебречь, при этом не снизив ценность тест-кейса.

**Отличия Test Suite от Test Scenario?**

Test Suite — это серия логически связанных Test case, которые позволяют проверить один из вариантов сценария. Test Scenario - представляет собой некий пользовательский сценарий по тестированию некой функциональности. Что-то, что пользователь может захотеть сделать с вашей системой, и вы хотите это проверить. Сценарий может иметь один или несколько Test Suite.

**Какие отличия у плана тестирования и стратегии тестирования?**

* План тестирования — это документ, в котором описываются объем, цель и значение задачи тестирования ПО, тогда как стратегия тестирования описывает, как необходимо проводить тестирование.
* План тестирования используется на уровне проекта, тогда как стратегия тестирования используется на уровне организации.
* План тестирования имеет первостепенную цель, состоящую в том, как тестировать, когда тестировать, и кто будет проверять, в то время как стратегия тестирования имеет первостепенную цель — какой техники придерживаться и что проверять.
* План тестирования может быть изменен, тогда как стратегия тестирования не может быть изменена.
* План тестирования выполняется тест менеджером (Test Manager), а стратегия тестирования — менеджером проекта (Project Manager).

**Виды тест планов?**

Чаще всего на практике приходится сталкиваться со следующими видами тест планов:

* Мастер Тест План (Master Plan or Master Test Plan)
* Тест План (Test Plan), назовем его детальный тест план
* План Приемочных Испытаний (Product Acceptance Plan) - документ, описывающий набор действий, связанных с [приемочным тестированием](http://www.protesting.ru/testing/levels/acceptance.html) (стратегия, дата проведения, ответственные работники и т. д.)

Явное отличие Мастер Тест Плана от просто Тест Плана в том, что мастер тест план является более статичным в силу того, что содержит в себе высокоуровневую (High Level) информацию, которая не подвержена частому изменению в процессе тестирования и пересмотра требований. Сам же детальный тест план, который содержит более конкретную информацию по стратегии, видам тестировании, расписанию выполнения работ, является «живым» документом, который постоянно претерпевает изменения, отражающие реальное положение дел на проекте.

В повседневной жизни на проекте может быть один Мастер Тест План и несколько детальных тест планов, описывающих отдельные модули одного приложения.

Для увеличения ценности вашего тест плана рекомендуется проводить его периодическое рецензирование со стороны участников проектной группы:

* Ведущий тестировщик
* Тест менеджер (менеджер по качеству)
* Руководитель разработки
* Менеджер Проекта

Каждый из перечисленных участников проекта, перед утверждением, проведет рецензию и внесет свои комментарии и предложения, которые помогут сделать Ваш тест план более полным и качественным.

**Что является основой для подготовки плана приемки? (PAP — Product Acceptance Plan)**

Прежде всего, вам нужны данные из следующих областей, чтобы подготовиться к приемке. Они могут варьироваться от компании к компании и от проекта к проекту.

* Документ с требованиями: Этот документ определяет, что именно требуется в проекте с точки зрения клиента.
* Вклад клиента: это могут быть обсуждения, неформальные чаты, письма электронной почты и т. д.
* План проекта: план проекта от менеджера проекта (PM) также служит хорошим вкладом в завершение вашего приемочного теста.

**В чем разница между тест-кейсом и чек-листом?**

Сила тест-кейса в том, что в нем все расписано очень детально, и с помощью тест-кейсов тестировать сможет даже человек, который ни разу не видел тестируемое им приложение. Но все это сложно обновлять или изменять.

Сила чек-листа в том, что он простой. Там нет глубокой детализации, это просто памятка. Но тестировать приложение по чек-листу сразу, без подготовки, не понимая, что подразумевается под «Зачарджить ордер на бэкофисе» (это где? это как? это что? это откуда и куда?) — практически невозможно.

**В чем разница между тест-кейсами высокого уровня и низкого уровня?**

Тест-кейсы высокого уровня охватывают основную функциональность продукта, такую как стандартные бизнес-процессы.

Низкоуровневые тест-кейсы относятся к пользовательскому интерфейсу в приложении.

**Чем Test case отличается от сценария тестирования?**

Test case — это артефакт тестирования для проверки определенного flow с определенными входными значениями, предварительными условиями тестирования, ожидаемым выходным сигналом и постусловиями, подготовленными для охвата определенного поведения. Тестовый сценарий может иметь одну или несколько ассоциаций с тестовым набором, что означает, что он может включать несколько тестовых наборов.

**Что такое тест-анализ/основа теста? (Test Analysis/Test Basis)**

Процесс поиска артефактов теста для определения условий / Test case. Следовательно, это также называется тестовой базой. Чаще всего информация получается из опыта или следующих источников:

* SRS (спецификация требований к ПО)
* BRS (спецификация бизнес-требований)
* Функциональные проектные документы

**Что такое документ бизнес-требований (BRD)?**

BRD предоставляет полное бизнес-решение для проекта, включая документацию о потребностях и ожиданиях клиентов. BRD выполняет следующие задачи.

* Получить соглашение с заинтересованными сторонами.
* Обеспечить ясность бизнес-требований.
* Описывает решение, которое соответствует потребностям клиента / бизнеса.
* Определяет входные данные для следующей фазы проекта.

**Что вы знаете о требованиях (уровни/виды и т. д.)?**

Виды требований по уровню:

* Бизнес-требования — определяют назначение ПО, описываются в документе о видении (vision) и границах проекта (scope).
* Пользовательские требования — определяют набор пользовательских задач, которые должна решать программа, а также способы (сценарии) их решения в системе. Пользовательские требования могут выражаться в виде фраз утверждений, в виде [сценариев использования](https://ru.wikipedia.org/wiki/%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%B8%D0%B9_%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) use case), [пользовательских историй](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B5_%D0%B8%D1%81%D1%82%D0%BE%D1%80%D0%B8%D0%B8) ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) user stories), сценариев взаимодействия (scenario).
* Функциональный уровень (функции).

Виды требований по характеру:

1. Функциональные требования — что система должна делать. К функциональным требованиям относят (из первого вытекает следующее):

* Бизнес-требования. Что система система должна делать с точки зрения бизнеса. Слово «бизнес» в данном контексте ближе к слову «заказчик». Пример бизнес-требования: промо-сайт, привлекающий внимание определенной аудитории к определенной продукции компании.
* Пользовательские требования – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы. Эти требования часто представляют в виде вариантов использования (Use Cases). Иначе говоря, пользовательские требования — это что может сделать пользователь: зарегистрироваться, посмотреть определенную информацию, пересчитать данные по определенному алгоритму и прочее.
* Функциональные требования – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований. Другими словами, что будут делать разработчики, чтобы выполнить пользовательские требования.

В группу функциональных требований относят и системные требования. Эти характеристики могут описывать требования как к аппаратному обеспечению (тип и частота процессора, объём оперативной памяти, объём жесткого диска), так и к программному окружению (операционная система, наличие установленных системных компонентов и сервисов и т. п.). Обычно такие требования составляются производителем или автором ПО. Например, для игры это могут быть требования такого типа: видеокарта — объём памяти от 64 Мб, совместимость сDirectX 9.0b и новейшие драйвера. Для сайта: ОС - Windows не ниже XP, браузеры IE не ниже 7.0 и так далее.

Почему важно указывать системные требования и утверждать их у заказчика? Если не указать, например, что важно обеспечить просмотр сайта в IE 6, то разработчики вполне могут выбрать такое архитектурное решение, которое не позволит корректно отображать сайт. Системные требования напрямую зависят от целевой аудитории проекта.

1. Нефункциональные требования. Иначе говоря, как будет работать система и почему именно так.

* Бизнес-правила. Они определяют почему система работать должна именно так, как написано. Это могут быть ссылки на законодательство, внутренние правила заказчика и прочие причины. Часто упускают этот раздел и получается, что некоторые системные решения выглядят нетипичным и совсем неочевидными. Например, многие табачные компании и компании, производящие алкоголь, требуют постоянного доказательства того, что промо-сайтами пользуются люди, достигшие определенного возраста. Это бизнес-правило (подтверждение возраста) возникает по требованию этических комитетов заказчика, хотя и несколько противоречит маркетинговым целям и требованиям по usability.
* Внешние интерфейсы. Это не только интерфейсы пользователя, но и протоколы взаимодействия с другими системами. Например, часто сайты связаны с CRM системами. Особенности протокола взаимодействия «сайт-CRM» также относятся к нефункциональным требованиям.
* Атрибуты качества. Атрибуты касаются вопросов прозрачности взаимодействия с другими системами, целостности, устойчивости и т.п. К таким характеристикам относятся:
  + легкость и простота использования (usability)
  + производительность (performance)
  + удобство эксплуатации и технического обслуживания (maintainability)
  + надежность и устойчивость к сбоям (reliability)
  + взаимодействия системы с внешним миром (interfaces)
  + расширяемость (scalability)
  + требования к пользовательским и программным интерфейсам (user and software interface).

Еще одна классификация:

* Явные требования: все, что вы написали. Чаще всего встречаются в документах, переданных заинтересованными сторонами команде разработчиков. Они могут принимать форму сложной спецификации дизайна, набора критериев приёма или описание каркаса ПО.
* Неявные требования — это второй тип. Это все то, что ожидают пользователи и что не было прописано в явных требованиях. Примеры включают производительность, удобство использования, доступность и безопасность. Рассмотрим облачный продукт хранения, который позволяет хранить ваши файлы в Интернете. Продукт получает новое явное требование: пользователи должны иметь доступ к частному контенту других пользователей через URL, используя кнопку совместного доступа.
* Скрытые требования: все, что будет приятным сюрпризом для клиента. Скрытые требования представляют собой функции, которые пользователи не ожидают увидеть в используемом продукте, основываясь на своем предыдущем опыте, но при их наличии данное ПО будет выигрывать в сравнении с конкурентами.

Источники требований:

* Федеральное и муниципальное отраслевое законодательство (конституция, законы, распоряжения)
* Нормативное обеспечение организации (регламенты, положения, уставы, приказы)
* Текущая организация деятельности объекта автоматизации
* Модели деятельности (диаграммы бизнес-процессов)
* Представления и ожидания потребителей и пользователей системы
* Журналы использования существующих программно-аппаратных систем
* Конкурирующие программные продукты

Методы выявления требований:

* Интервью, опросы, анкетирование
* [Мозговой штурм](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B3%D0%BE_%D1%88%D1%82%D1%83%D1%80%D0%BC%D0%B0), семинар
* Наблюдение за производственной деятельностью, «фотографирование» рабочего дня
* Анализ нормативной документации
* Анализ моделей деятельности
* Анализ конкурентных продуктов
* Анализ статистики использования предыдущих версий системы

Требования начала для производства ПО:

* необходимое тестовое окружение,
* билд/ресурс/предмет тестирования,
* код, БД, прочие компоненты объекта тестирования «заморожены», т. е. не изменяются в период всей сессии тестирования,
* модификация требований (хотя бы прямых) «заморожена»,
* известно направление тестирования,
* известны сроки на сессию тестирования.

Есть и другие условия, но они менее значимы и сильно зависят от конкретного процесса в компании.

**Рассажите, какие есть требования к самим требованиям?**

* Понятность. Если требования понятны кому-то одному, но не понятны всем остальным участникам, или наоборот понятны всем кроме одного — свидетельство об ошибке в их составлении. Как понять, что требование понятно?
* Реализуемость. Требования должны быть реализуемы в рамках заявленных платформ вообще, а также реализуемыми в заявленные сроки. Разработчик должен в первую очередь смотреть на этот атрибут при проведении ревью.
* Обращайте внимание на очень большие и очень маленькие числа в требованиях. Реализовать отклик в 1 милисекунду или закачку файлов в 100ГБ, наверное, можно, только зачем, и будет ли это соответствовать основным функциональным задачам сайта?
* Полнота. Зависит от назначения и формата требований. В общем случае требования должны полностью описывать то, что в них изначально подразумевается (не должно быть недосказанности или «и т. д. »)
* Непротиворечивость. Требования должны быть поняты однозначно. Это «тест на внимательность» для тестировщика. Хорошо, если противоречие содержится в двух требованиях, расположенных рядом. Но в длинных документах требования могут занимать несколько страниц, и требования на 15 странице могут противоречить требованиям со 2 страницы. Так же требования должны не противоречить законам физики, геометрии, математики и прочим обусловленными внешними законами и обстоятельствами
* Измеримость. Требования можно посчитать и измерить (нет — «большая база», «быстрый отклик»)
* Актуальность. Требования не устарели.
* Проверяемость. Реализованность требования может быть определена через один из четырёх возможных методов: осмотр, демонстрация, тест или анализ.

**Дефекты и ошибки**

**Что такое дефект?**

Разница между ожидаемым и фактическим результатом.

**Классы дефектов?**

* Функциональные
  + Дефекты требований
  + Дефекты функций
  + Функциональные дефекты самого процесса тестирования
* Системные
  + Внутренние взаимодействия (интерфейсы)
  + Аппаратная часть
  + ОС
  + Архитектура ПО
* Дефекты, связанные с процессом
  + Арифметика (например, нюансы с округлениями в коде)
  + Инициализация
  + Последовательность
  + Статическая логика (например, валидация форм)
* Дефекты, связанные с данными
* Дефекты, связанные со стандартами
* Дефекты пользовательского интерфейса (UI)

**Какие есть категории дефектов?**

Существует три основных категории дефектов:

* Wrong: это указывает на несоответствие в требовании и реализации. Это подразумевает отклонение от данной спецификации.
* Missing: конечный продукт не имеет функции, соответствующей требованию. Это отличается от спецификаций и означает, что вы неправильно задокументировали требование.
* Extra: вы добавили функцию, которую клиент не запрашивал. Это снова отклонение от спецификации. И пользователям продукта может понравиться эта функция. Но это все еще дефект, потому что это не часть спецификаций.

**Error/Mistake/Defect/Bug/Failure/Fault?**

В русском языке переводятся практически одинаково, но это разные термины. В некоторых источниках данные термины следуют за категорями, описанными выше.

* Ошибка (Error) возникает из-за ошибки (Mistake) в написании кода разработчиком.
* Дефект (Defect) это скрытый недостаток в ПО, возникший из-за ошибки в написании кода.
* Когда дефект (Defect) обнаруживается тестировщиком, он называется багом (Bug).
* Если тестировщики упустили дефект и его нашёл пользователь, то это сбой (Failure).
* Если программа в итоге не выполняет свою функцию, то это отказ (Fault).

**Каково содержание эффективного сообщения об ошибке?**

* Project
* Subject
* Description
* Summary
* Detected By (Name of the Software Tester)
* Assigned To (Name of the Developer of the feature)
* Test Lead (Name)
* Detected in Version
* Closed in Version
* Date Detected
* Expected Date of Closure
* Actual Date of Closure
* Priority (Medium, Low, High, or Urgent)
* Severity (Range => 1 to 5)
* Status.
* Bug ID
* Attachment
* Test case Failed (Total no. of Test cases which are failing for a Bug)

**Несколько ключевых моментов, которые следует учитывать при написании отчета об ошибке?**

* Воспроизведите ошибку 2-3 раза.
* Используйте некоторые ключевые слова, связанные с вашей ошибкой, и выполните поиск в инструменте отслеживания дефектов.
* Проверьте в аналогичных модулях.
* Сообщите о проблеме немедленно.
* Напишите подробные шаги для воспроизведения ошибки.
* Напишите хорошее summary дефекта. Следите за словами в процессе написания сообщения об ошибке, они не должны оскорблять людей. Никогда не используйте капс, объясняя проблему.
* Желательно проиллюстрировать проблему с помощью правильных скриншотов.
* Перед публикацией перепроверьте два или три раза ваш отчет об ошибке.

**Серьезность и Приоритет Дефекта (Severity & Priority)**

Разные системы баг трекинга предлагают нам разные пути описания серьезности и приоритета баг репорта, неизменным остается лишь смысл, вкладываемый эти поля. Все знают такой баг-трекер, как Atlassian JIRA. В нем, начиная с какой-то версии вместо одновременного использования полей Severity и Priority, оставили только Priority, которое собрало в себе свойства обоих полей. Но, всё же, разделение этих понятий может быть очень важно, а точнее использование обоих полей Severity и Priority, так как смысл, вкладываемый в них, различный:

Серьезность — представляет серьезность / глубину ошибки в контексте работоспособности самого ПО. Приоритет - указывает на очередность выполнения задачи или устранения дефекта.

Серьезность — Описывает точку зрения приложения. Приоритет - точку зрения бизнеса.

пример: я в ходе тестирования нашёл дефект, довольно критичный для приложения на мой взгляд т.к. этот дефект закрывает доступ к 20% функционала. ставлю такому багу сиверити «критикал». ПМ видит багрепорт, анализирует ситуацию (поджимающие сроки, этот функционал будет рефакториться или вообще выкидываться в следующей итерации и т.п.) и ставит приорити — «медиум» или ниже. а девелопер при работе с баг- тасктреккером руководствуется исключительно приорити, т.к. приорити и существует для регулирования очерёдности выполнения тасков.

Градация Серьезности дефекта (Severity):

* Блокирующий (S1 – Blocker) – тестирование значительной части функциональности вообще недоступно. Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна.
* Критический (S2 – Critical) – Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, то есть не работает важная часть одной какой-либо функции либо не работает значительная часть, но имеется workaround (обходной путь/другие входные точки), позволяющий продолжить тестирование.
* Значительный (S3 – Major)– не работает важная часть одной какой-либо функции/бизнес-логики, но при выполнении специфических условий, либо есть workaround, позволяющий продолжить ее тестирование либо не работает не очень значительная часть какой-либо функции. Также относится к дефектам с высокими visibility – обычно не сильно влияющие на функциональность дефекты дизайна, которые, однако, сразу бросаются в глаза.
* Незначительный (S4 – Minor) – часто ошибки GUI, которые не влияют на функциональность, но портят юзабилити или внешний вид. Также незначительные функциональные дефекты, либо которые воспроизводятся на определённом устройстве.
* Тривиальный (S5 – Trivial) – почти всегда дефекты на GUI — опечатки в тексте, несоответствие шрифта и оттенка и т.п., либо плохо воспроизводимая ошибка, не касающаяся бизнес-логики, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Серьезность ошибки может быть низкой, средней или высокой, в зависимости от контекста.

* Дефект интерфейса пользователя — Низкая
* Пограничные дефекты — Средняя
* Обработка ошибок — Средняя
* Дефекты расчета — Высокая
* Неверно истолкованные данные — Высокая
* Аппаратные сбои — Высокий
* Проблемы совместимости — Высокая
* Дефекты потока управления — Высокая
* Условия нагрузки (утечки памяти при нагрузочном тестировании) – Высокая

Градация Приоритета дефекта (Priority):

* P1 Высокий (High) Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критичным для проекта.
* P2 Средний (Medium) Ошибка должна быть исправлена, ее наличие не является критичным, но требует обязательного решения.
* P3 Низкий (Low) Ошибка должна быть исправлена, но ее наличие не является критичным и не требует срочного решения.

**Может ли быть высокий severity и низкий priority? А наоборот?**

Да, такое может быть при некоторых стечениях обстоятельств.

* Очень низкая серьезность с высоким приоритетом: ошибка логотипа для любого продающего веб-сайта может иметь низкую серьезность, поскольку она не повлияет на функциональность веб-сайта, но может иметь высокий приоритет, поскольку вы не хотите, чтобы дальнейшая продажа продолжалась с неправильным логотипом. Или, например, сайт-визитка, показывающий только основную краткую информацию об организации может содержать грамматические ошибки, которые в иных случаях были бы с минимальным приоритетом, но в данном приведут к репутационным потерям.
* Очень высокая серьезность с низким приоритетом. Аналогичным образом, для веб-сайта авиакомпании дефект функциональности бронирования может быть серьезным, но может иметь низкий приоритет, так как исправление уже может быть запланировано в следующем цикле и возможно бронирование по телефону.

**Жизненный цикл дефекта?**

Жизненный цикл дефекта — это представление различных состояний дефекта, в которых он пребывает от начального до конечно этапа своего существования. Он может варьироваться от компании к компании или даже настраиваться для некоторых проектов.

Допустим вы нашли баг и зарегистрировали его в баг трекинг системе. Согласно нашей блок-схеме он получит статус «Новый». Тестировщик, ответственный за валидацию новых баг репортов, или координатор проекта (в зависимости от распределения ролей в вашей команде) может перевести его в один из следующих статусов:

«Отклонен», если данный баг невалидный или повторный, или же его просто не смогли воспроизвести

«Отсрочен», если данный баг не нужно исправлять в данной итерации

«Открыт», если исправление бага необходимо

Рассмотрим теперь по порядку каждый из вариантов.

Отклонен. В этом случае вы можете либо поспорить о судьбе вашего багрепорта, изменив статус на «Переоткрыт» либо закрыть его — статус «Закрыт»

Отсрочен. Баг репорт в статусе «Отсрочен» можно перевести в статус «Открыт», когда потребуется исправление либо в статус «Закрыт», если уже не потребуется.

Открыт. Именно в таком состоянии разработчик получает баг репорт для исправления. Он может отклонить (дальнейшие действия смотрите в пункте 1) или исправить баг. Баг репорт в статусе «Исправлен» переводится на тестировщика для проверки. В случае если проблема все еще воспроизводится, выставляется статус «Переоткрыт» и баг репорт направляется назад на доработку к разработчику. Если же исправление было успешным, то баг репорт переводится в статус «Закрыт».

Хотим отметить, что данная схема сильно упрощена. Для большей наглядности и, возможно, удобства работы на проекте, вы можете добавить дополнительные статусы и переходы, тем более, что современные баг трекинговые системы позволяют это делать. Правда имейте ввиду, что излишне запутанные схемы переходов и лишние статусы могут значительно усложнить жизнь.

Примечание 1: в некоторых системах баг трекинга, созданный баг репорт сразу получает статус «Открыт» без дополнительной валидации

Примечание 2: многие баг трекинговые системы позволяют переоткрывать закрытые баги, однако лично я против такой практики, поэтому и не описывал подобный переход в выше представленном жизненном цикле

Примечание 3: Рассмотренный выше жизненный цикл основан на том, что в команде есть кто-то, ответственный за назначение баг репортов. В случае, если такой роли на проекте нет, то баги назначаются разработчиками самостоятельно, и тогда во избежании путанницы, есть смысл ввести еще один промежуточный статус «В разработке» (In progress), показывающий, что данный баг репорт уже назначен и находится на стадии исправления.

**Пришёл баг из продакшена, что делаем?**

Воспроизводим, запускаем по пути жизненного цикла дефекта и анализируем причины, как данный дефект прошёл в прод. После исправления дефекта разработчиком проводим повторное тестирование. Добавляем данный дефект в регрессию. В зависимости от охватываемого функционала и Root Cause этих багов принимается решение о проведении санитарного/регрессионного тестирования после подтверждающего.

**Что такое утечка дефектов и релиз бага? (Bug Leackage & Bug Release)**

Релиз бага — это когда программное обеспечение или приложение передается группе тестирования, зная, что дефект присутствует в выпуске. При этом приоритет и серьезность ошибки низки, поскольку ошибка может быть удалена до окончательной передачи обслуживания.

Утечка бага — когда баг обнаруживается конечными пользователями или заказчиком, а не обнаруживается группой тестирования во время тестирования программного обеспечения.

**Что означает плотность дефектов при тестировании ПО?**

Плотность дефектов — это количество дефектов, подтвержденных в программном обеспечении / модуле в течение определенного периода эксплуатации или разработки, деленное на размер ПО / модуля. Это позволяет решить, готова ли часть ПО к выпуску. Плотность дефектов рассчитывается на тысячу строк кода, и обозначается KLOC.

Однако не существует фиксированного стандарта для плотности ошибок, исследования показывают, что один дефект на тысячу строк кода обычно считается признаком хорошего качества проекта.

**Что означает процент обнаружения дефектов при тестировании ПО?**

Процент обнаружения дефектов (DDP) — это тип метрики тестирования. Он показывает эффективность процесса тестирования путем измерения соотношения дефектов, обнаруженных до выпуска и сообщенных после выпуска клиентами. Например, скажем, QA зарегистрировало 70 дефектов во время цикла тестирования, а клиент сообщил еще 20 после выпуска. DDP составит 72,1% после расчета 70 / (70 + 20) = 72,1%.

**Что означает эффективность устранения дефектов при тестировании ПО? (DRP)**

Эффективность устранения дефектов (DRP) — это тип метрики тестирования. Это показатель эффективности команды разработчиков для устранения проблем перед выпуском. Он измеряется как отношение зафиксированных дефектов к общему количеству обнаруженных проблем. Например, допустим, что во время цикла тестирования было обнаружено 75 дефектов, в то время как 62 из них были устранены командой разработчиков во время измерения. DRE достигнет 82,6% после расчета 62/75 = 82,6%.

**Что означает эффективность Test case в тестировании ПО? (TCE)**

Эффективность тестирования (TCE) — это тип метрики тестирования. Это четкий показатель эффективности выполнения Test case на этапе выполнения теста в выпуске. Это помогает в обеспечении и измерении качества Test case. Эффективность тестового набора (TCE) => (Количество обнаруженных дефектов / Количество выполненных Test case) \* 100

**Возраст дефекта в тестировании ПО?**

Возраст дефекта — это время, прошедшее между днем обнаружения тестером и днем, когда разработчик исправил его.

**Что такое принцип Парето в тестировании ПО?**

В тестировании ПО принцип Парето говорит о том, что 80% всех ошибок приходится на 20% кода.

**Каковы различные способы применения принципа Парето в тестировании ПО?**

1. Расставьте дефекты по их причинам, а не по последствиям. Не клубите ошибки, которые дают тот же результат. Группируйте проблемы в зависимости от того, в каком модуле они возникают.

1. Сотрудничайте с командой разработчиков, чтобы найти новые способы классификации проблем. Например, используйте ту же статическую библиотеку для компонентов, которые учитывают большинство ошибок.
2. Больше энергии вкладывайте в поиск проблемных областей в исходном коде, а не в случайный поиск.
3. Переупорядочьте Test case и выберите наиболее важные для начала.
4. Обратите внимание на реакцию конечного пользователя и оцените зоны риска.

**В чем основное отличие отладки от тестирования? (Debugging Vs. Testing)**

Тестирование заключается в обнаружении дефектов при использовании продукта, а отладка — в части кода, вызывающей сбой. Отладка изолирует проблемную область в коде, сделанном разработчиком, тогда как Тестирование идентифицирует ошибку в приложении и выполняется тестером.

**Почему в программном обеспечении есть ошибки?**

* Недопонимание.
* Ошибки программирования.
* Сжатые сроки.
* Изменение в требованиях.
* Сложность ПО.

**Что вы будете делать, если во время тестирования появится ошибка?**

Когда обнаруживается ошибка, запустите больше тестов, чтобы убедиться, что проблема имеет четкое описание. Запустите еще несколько тестов, чтобы убедиться, что одна и та же проблема не существует с разными входами. Как только мы полностью уверены в ошибке, мы можем добавить подробности и сообщить о ней.

**Как вы справляетесь с невоспроизводимой ошибкой?**

Следующие типы ошибок относятся к категории невоспроизводимых.

* Наблюдаемые дефекты из-за недостатка памяти
* Проблемы, возникающие из-за адреса, указывающего на область памяти, которая не существует.
* Состояние гонки — ошибка проектирования [многопоточной](https://ru.wikipedia.org/wiki/%D0%9C%D0%BD%D0%BE%D0%B3%D0%BE%D0%BF%D0%BE%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D1%8C) системы или приложения, при которой работа системы или приложения зависит от того, в каком порядке выполняются части кода

Тестировщик может предпринять следующие действия для устранения невоспроизводимых ошибок.

* Выполните шаги теста, близкие к описанию ошибки.
* Оцените тестовую среду.
* Изучите и оцените результаты выполнения теста.
* Держите ресурсы и временные ограничения под контролем.

**Если продукт находится в производстве и один из его модулей обновляется, то необходимо ли провести повторную проверку?**

Повторная проверка относится только к исправленным дефектам. После обновления модуля желательно выполнить регрессионное тестирование и запустить интеграционные тесты и прочие тесты для всех других модулей. После, QA следует провести Системное тестирование. Всё зависит от компании и бюджета, стратегии/политики тестирования или политики качества.

**Что такое анализ рисков?**

Анализ рисков — это метод выявления вещей, которые могут пойти не так в проекте разработки ПО. Они могут негативно повлиять на объем, качество, своевременность и стоимость проекта. Тем не менее, все участники проекта участвуют в минимизации риска. Но именно лидер гарантирует, что вся команда понимает индивидуальную роль в управлении риском.

Как выполнять анализ рисков во время тестирования ПО? Анализ рисков — это процесс выявления скрытых проблем, которые могут помешать успешной доставке приложения. Он также устанавливает приоритетность последовательности устранения выявленных рисков для целей тестирования. Ниже приведены некоторые из рисков, которые представляют интерес для обеспечения качества.

* Новое оборудование
* Новые технологии
* Новый инструмент автоматизации
* Последовательность доставки кода
* Наличие тестовых ресурсов для приложения

Мы выделяем их в три категории, которые заключаются в следующем.

* Высокая важность: влияние ошибки значительно на другие функциональные возможности приложения
* Средний: это несколько терпимо, но не желательно.
* Низкий: терпимо. Этот вид риска не влияет на бизнес компании.

**В чем разница между coupling и cohesion?**

Сцепление/связность (Cohesion) - это степень, которая измеряет зависимость компонента ПО, который объединяет связанные функциональные возможности в одном блоке, тогда как соединение/связь (coupling) представляет его в другой группе.

Cohesion имеет дело с функциональностью, которая связана с различными процессами в пределах одного модуля, в то время как coupling имеет дело с тем, насколько один модуль зависит от других модулей в продукте.

Это хорошо для увеличения Cohesion между программным обеспечением, тогда как coupling не рекомендуется.

**Что такое скрытый дефект? (Latent defect)**

Этот дефект является существующим дефектом в системе, но он не вызывает никаких сбоев, поскольку точный набор условий еще никогда не выполнялся.

**Что такое маскировка ошибок, объясните примером?**

Когда наличие одного дефекта скрывает наличие другого дефекта в системе, это называется маскированием неисправностей. Пример: если «отрицательное значение» вызывает исключение необработанного системного исключения, разработчик предотвратит ввод отрицательных значений. Это решит проблему и скроет дефект обработки необработанных исключений.

**Категории отладки? (Debugging)**

* Отладка грубой силой (Brute force debugging)
* Откат
* Устранение причины
* Нарезка программы (Program Slicing)
* Анализ дерева отказов (Fault tree analysis)

**Что такое Эффективность удаления дефектов? (DRE — Defect Removal Efficiency)**

Этот показатель используется для измерения эффективности теста. Из этого показателя мы узнаем, сколько ошибок мы обнаружили в наборе Test case. Формула для расчета DRE имеет вид

DRE = Количество ошибок во время тестирования / (количество ошибок во время тестирования + количество ошибок, обнаруженных пользователем)

**Что такое сортировка дефектов? (Bug triage)**

Это формальный процесс, позволяющий определить, какие ошибки являются важными, путем определения их приоритетов на основе их серьезности, частоты, риска и других важных параметров. Тестеры назначают приоритет (высокий, средний, низкий) каждой ошибке на bug triage meeting и в зависимости от приоритета эти ошибки будут исправлены в соответствующем порядке. Делая это, мы экономим ресурсы организации.

**SDLC и STLC**

**Что вы знаете о жизненном цикле разработки ПО? (SDLC — Software Development Lifecycle)**

SDLC определяет все стандартные фазы, которые участвуют в процессе разработки ПО. Жизненный цикл SDLC — это процесс поэтапной разработки ПО, которому следуют в программных проектах. Каждый этап SDLC производит результаты, необходимые для следующего этапа жизненного цикла. Требования транслируются в дизайн. Код пишется в соответствии с дизайном. Тестирование должно проводиться на разработанном продукте в соответствии с требованиями. Развертывание должно быть сделано после завершения тестирования.

* Этап требований (Requirement Phase): Сбор и анализ требований является наиболее важной фазой в жизненном цикле разработки программного обеспечения. Бизнес-аналитик получает требование от Заказчика / Клиента в соответствии с бизнес-потребностями и документирует требования в Спецификации бизнес-требований (Business Requirement Specification, название документа зависит от Организации. Некоторые примеры: Спецификация требований клиента (CRS — Customer Requirement Specification), Бизнес-спецификация (BS - Business Specification), и т. д. ), и предоставляет то же самое для команды разработчиков.
* Этап анализа (Analysis Phase): После сбора и анализа требований следующим шагом является определение и документирование требований к продукту и их утверждение заказчиком. Это делается с помощью документа Спецификация требований к программному обеспечению (SRS — Software Requirement Specification). SRS состоит из всех требований к продукту, которые должны быть спроектированы и разработаны в течение жизненного цикла проекта. Ключевыми людьми, вовлеченными в этот этап, являются Project Manager, Business Analyst и Senior члены команды. Результатом этого этапа является спецификация требований к программному обеспечению.
* Этап проектирования (Design Phase): высокоуровневый дизайн (HLD — High-Level Design) – это про архитектуру программного продукта, который должен быть разработан, и выполняется архитекторами и старшими разработчиками. Низкоуровневый дизайн (LLD - Low-Level Design) - это делается старшими разработчиками. Он описывает, как должна работать каждая функция продукта и как должен работать каждый компонент. Здесь будет только дизайн, а не код. Результатом этого этапа является документ высокого уровня и документ низкого уровня, который служит входными данными для следующего этапа.
* Этап разработки (Development Phase): Разработчики всех уровней участвуют в этом этапе. На этом этапе мы начинаем создавать программное обеспечение и начинаем писать код для продукта. Результатом этого этапа является исходный код документа (SCD — Source Code Document) и разработанный продукт.
* Этап тестирования (Testing Phase): Когда программное обеспечение готово, оно отправляется в отдел тестирования, где группа тестирования тщательно тестирует его на наличие различных дефектов. Они либо тестируют программное обеспечение вручную, либо используют инструменты автоматического тестирования, в зависимости от процесса, определенного в STLC (жизненный цикл тестирования программного обеспечения), и гарантируют, что каждый компонент программного обеспечения работает нормально. Как только QA удостоверится, что программное обеспечение не содержит серьезных ошибок, оно переходит к следующему этапу, который является реализацией. Результатом этого этапа является качество продукта и артефакты тестирования.
* Этап развертывания и обслуживания (Deployment and Maintenance Phase): После успешного тестирования продукт доставляется / развертывается заказчику для использования. Развертывание выполняется Deployment/Implementation engineers. Однажды, когда клиенты начнут использовать разработанную систему, возникнут реальные проблемы, и время от времени их придется решать. Устранение проблем, обнаруженных заказчиком, происходит на этапе обслуживания. Техническое обслуживание должно выполняться в соответствии Соглашением об уровне обслуживания (SLA — Service Level Agreement)

**Что такое цикл/колесо Деминга? (Deming circle/cycle/wheel)**

Цикл PDCA (Plan – Do – Check – Act) — это непрерывный (до получения конечного результата) итеративный четырехступенчатый метод управления, используемый в бизнесе для постоянного улучшения процессов. Это одна из ключевых концепций качества, и она также называется кругом/ циклом/ колесом Деминга.

* Plan: Запланируйте изменения (либо для решения проблемы, либо для улучшения некоторых областей) и решите, какую цель достичь. Здесь мы определяем цель, стратегию и вспомогательные методы для достижения цели нашего плана.
* Do: Разработать или пересмотреть бизнес-требования в соответствии с планом. Здесь мы реализуем план (с точки зрения реализации плана) и проверяем его эффективность
* Check: Оцените результаты, чтобы убедиться, что мы достигнем целей, как планировалось. Здесь мы составляем контрольный список для записи того, что прошло хорошо, а что не сработало (извлеченные уроки)
* Act: Если изменения не соответствуют запланированным, продолжайте цикл для достижения цели с другим планом. Здесь мы принимаем меры по факту того, что не работает, как планировалось. Задача состоит в том, чтобы продолжать пытаться улучшить процесс с другим планом.

**Модели разработки ПО?**

Waterfall (каскадная модель, или «водопад»). В этой модели разработка осуществляется поэтапно: каждая следующая стадия начинается только после того, как заканчивается предыдущая. Если всё делать правильно, «водопад» будет наиболее быстрой и простой моделью. Применяется уже почти полвека, с 1970-х.

Преимущества «водопада»: Разработку просто контролировать. Заказчик всегда знает, чем сейчас заняты программисты, может управлять сроками и стоимостью. Стоимость проекта определяется на начальном этапе. Все шаги запланированы уже на этапе согласования договора, ПО пишется непрерывно «от и до».

Не нужно нанимать тестировщиков с серьёзной технической подготовкой. Тестировщики смогут опираться на подробную техническую документацию.

Недостатки каскадной модели: Тестирование начинается на последних этапах разработки. Если в требованиях к продукту была допущена ошибка, то исправить её будет стоить дорого. Тестировщики обнаружат её, когда разработчик уже написал код, а технические писатели — документацию.

Заказчик видит готовый продукт в конце разработки и только тогда может дать обратную связь. Велика вероятность, что результат его не устроит. Разработчики пишут много технической документации, что задерживает работы. Чем обширнее документация у проекта, тем больше изменений нужно вносить и дольше их согласовывать.

«Водопад» подходит для разработки проектов в медицинской и космической отрасли, где уже сформирована обширная база документов (СНиПов и спецификаций), на основе которых можно написать требования к новому ПО. При работе с каскадной моделью основная задача — написать подробные требования к разработке. На этапе тестирования не должно выясниться, что в них есть ошибка, которая влияет на весь продукт.

V-образная модель (разработка через тестирование)

Это усовершенствованная каскадная модель, в которой заказчик с командой программистов одновременно составляют требования к системе и описывают, как будут тестировать её на каждом этапе. История этой модели начинается в 1980-х.

Преимущества V-образной модели: Количество ошибок в архитектуре ПО сводится к минимуму.

Недостатки V-образной модели: Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде».

V-модель подходит для проектов, в которых важна надёжность и цена ошибки очень высока. Например, при разработке подушек безопасности для автомобилей или систем наблюдения за пациентами в клиниках.

Incremental Model (инкрементная модель)

Это модель разработки по частям (increment в переводе с англ. — приращение) уходит корнями в 1930-е. Рассмотрим её на примере создания социальной сети.

Заказчик решил, что хочет запустить соцсеть, и написал подробное техническое задание. Программисты предложили реализовать основные функции — страницу с личной информацией и чат. А затем протестировать на пользователях, «взлетит или нет».

Команда разработки показывает продукт заказчику и выпускает его на рынок. Если и заказчику, и пользователям социальная сеть нравится, работа над ней продолжается, но уже по частям.

Программисты параллельно создают функциональность для загрузки фотографий, обмена документами, прослушивания музыки и других действий, согласованных с заказчиком. Инкремент за инкрементом они совершенствуют продукт, приближаясь к описанному в техническом задании.

Преимущества инкрементной модели

Не нужно вкладывать много денег на начальном этапе. Заказчик оплачивает создание основных функций, получает продукт, «выкатывает» его на рынок — и по итогам обратной связи решает, продолжать ли разработку. Можно быстро получить фидбэк от пользователей и оперативно обновить техническое задание. Так снижается риск создать продукт, который никому не нужен. Ошибка обходится дешевле. Если при разработке архитектуры была допущена ошибка, то исправить её будет стоить не так дорого, как в «водопаде» или V-образной модели.

Недостатки инкрементной модели: Каждая команда программистов разрабатывает свою функциональность и может реализовать интерфейс продукта по-своему. Чтобы этого не произошло, важно на этапе обсуждения техзадания объяснить, каким он будет, чтобы у всех участников проекта сложилось единое понимание.

Разработчики будут оттягивать доработку основной функциональности и «пилить мелочёвку». Чтобы этого не случилось, менеджер проекта должен контролировать, чем занимается каждая команда.

Инкрементная модель подходит для проектов, в которых точное техзадание прописано уже на старте, а продукт должен быстро выйти на рынок.

Iterative Model (итеративная модель)

Это модель, при которой заказчик не обязан понимать, какой продукт хочет получить в итоге, и может не прописывать сразу подробное техзадание.

Рассмотрим на примере создания мессенджера, как эта модель работает.

Заказчик решил, что хочет создать мессенджер. Разработчики сделали приложение, в котором можно добавить друга и запустить чат на двоих.

Мессенджер «выкатили» в магазин приложений, пользователи начали его скачивать и активно использовать. Заказчик понял, что продукт пользуется популярностью, и решил его доработать.

Программисты добавили в мессенджер возможность просмотра видео, загрузки фотографий, записи аудиосообщений. Они постепенно улучшают функциональность приложения, адаптируют его к требованиям рынка.

Преимущества итеративной модели: Быстрый выпуск минимального продукта даёт возможность оперативно получать обратную связь от заказчика и пользователей. А значит, фокусироваться на наиболее важных функциях ПО и улучшать их в соответствии с требованиями рынка и пожеланиями клиента. Постоянное тестирование пользователями позволяет быстро обнаруживать и устранять ошибки.

Недостатки итеративной модели: Использование на начальном этапе баз данных или серверов — первые сложно масштабировать, а вторые не выдерживают нагрузку. Возможно, придётся переписывать большую часть приложения. Отсутствие фиксированного бюджета и сроков. Заказчик не знает, как выглядит конечная цель и когда закончится разработка.

Итеративная модель подходит для работы над большими проектами с неопределёнными требованиями, либо для задач с инновационным подходом, когда заказчик не уверен в результате.

Spiral Model (спиральная модель)

Используя эту модель, заказчик и команда разработчиков серьёзно анализируют риски проекта и выполняют его итерациями. Последующая стадия основывается на предыдущей, а в конце каждого витка — цикла итераций — принимается решение на основе рисков, продолжать ли развивать проект.

Рассмотрим, как функционирует эта модель, на примере разработки системы «Умный дом».

Заказчик решил, что хочет сделать такую систему, и заказал программистам реализовать управление чайником с телефона. Они начали действовать по модели «водопад»: выслушали идею, провели анализ предложений на рынке, обсудили с заказчиком архитектуру системы, решили, как будут её реализовывать, разработали, протестировали и «выкатили» конечный продукт.

Заказчик оценил результат и риски: насколько нужна пользователям следующая версия продукта — уже с управлением телевизором. Рассчитал сроки, бюджет и заказал разработку. Программисты действовали по каскадной модели и представили заказчику более сложный продукт, разработанный на базе первого.

Заказчик подумал, что пора создать функциональность для управления холодильником с телефона. Но, анализируя риски, понял, что в холодильник сложно встроить Wi-Fi-модуль, да и производители не заинтересованы в сотрудничестве по этому вопросу. Следовательно, риски превышают потенциальную выгоду. На основе полученных данных заказчик решил прекратить разработку и совершенствовать имеющуюся функциональность, чтобы со временем понять, как развивать систему «Умный дом».

Спиральная модель похожа на инкрементную, но здесь гораздо больше времени уделяется оценке рисков. С каждым новым витком спирали процесс усложняется. Эта модель часто используется в исследовательских проектах и там, где высоки риски.

Преимущества спиральной модели: Большое внимание уделяется проработке рисков.

Недостатки спиральной модели: Есть риск застрять на начальном этапе — бесконечно совершенствовать первую версию продукта и не продвинуться к следующим. Разработка длится долго и стоит дорого.

На основе итеративной модели была создана Agile — не модель и не методология, а скорее подход к разработке.

**Что такое Agile?**

В классической модели waterfall каждая стадия начиналась после предыдущей без возврата назад и только в самом конце начиналось тестирование. Можно ли обеспечить качество, когда уже всё готово? В книге «Как тестируют в Googe» говорится, что QA не отвечает единолично за качество продукта, за это отвечает вся команда и в первую очередь разработчики.

В результате post-development тестирования создавалась иллюзия качественного продукта, но это не обеспечение качества, а скорее QC. Еще одним следствием следования каскадной модели являлось то, что команда старалась реализовать сразу все требования и к этапу тестирования выяснялось, что требуется много доделок/переделок и в результате релиз откладывался. Помимо прочего, пока разработчики писали код, тестировщики бездействовали. Безусловно, что-то писалось по требованиям и интуитивно, но смысл понятен. Нередко из-за срыва срока релизов сокращался срок, отводимый на тестирование, что также неблагоприятно сказывалось на итоговом качестве продукта.

Далее вмете с прогрессом пошла эволюция процессов SDLC и пришло понимание необходимости встраивания процессов обеспечения качества в жизненный цикл разработки продукта. Таким образом появлялись новые модели разработки и однажды группой энтузиастов был придуман Agile-манифест — основной документ, содержащий описание ценностей и принципов гибкой разработки программного обеспечения (4 ценности, 12 принципов):

Ценности:

1. Люди и взаимодействие важнее процессов и инструментов.
2. Работающий продукт важнее исчерпывающей документации.
3. Сотрудничество с клиентом важнее согласования условий контракта.
4. Готовность к изменениям важнее следования первоначальному плану.

Основные принципы:

1. Наивысшим приоритетом является удовлетворение потребностей клиента, благодаря регулярной и ранней поставке ценного программного обеспечения.
2. Изменение требований приветствуется, даже на поздних стадиях разработки.
3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.
4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.
5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.
6. Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.
7. Работающий продукт — основной показатель прогресса.
8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно.
9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.
10. Простота — искусство минимизации лишней работы — крайне необходима.
11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.
12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Agile включает в себя практики, подходы и методологии, которые помогают создавать продукт более эффективно:

* экстремальное программирование (Extreme Programming, XP);
* бережливую разработку программного обеспечения (Lean);
* фреймворк для управления проектами Scrum;
* разработку, управляемую функциональностью (Feature-driven development, FDD);
* разработку через тестирование (Test-driven development, TDD);
* методологию «чистой комнаты» (Cleanroom Software Engineering);
* итеративно-инкрементальный метод разработки (OpenUP);
* методологию разработки Microsoft Solutions Framework (MSF);
* метод разработки динамических систем (Dynamic Systems Development Method, DSDM);
* метод управления разработкой Kanban.

Не всё перечисленное в списке — методологии. Например, Scrum чаще называют не методологией, а фреймворком. В чём разница? Фреймворк — это более сформированная методология со строгими правилами. В скраме все роли и процессы чётко прописаны. Помимо Scrum, часто используют Kanban. Сегодня это одна из наиболее популярных методологий разработки ПО. Команда ведёт работу с помощью виртуальной доски, которая разбита на этапы проекта. Каждый участник видит, какие задачи находятся в работе, какие — застряли на одном из этапов, а какие уже дошли до его столбца и требуют внимания.

В отличие от скрама, в канбане можно взять срочные задачи в разработку сразу, не дожидаясь начала следующего спринта. Канбан удобно использовать не только в работе, но и в личных целях — распределять собственные планы или задачи семьи на выходные, наглядно отслеживать прогресс.

(Красный – waterfall, синий – разработка по гибкой методологии)

Впоследствии был разработан отдельный Манифест тестирования в Agile:

* постоянное тестирование, а не только в конце разработки
* предотвращение багов более значимо, чем их поиск
* понимание тестируемого продукта выше проверки функционала
* построение лучшей системы в связке с командой выше поиска методов ее сломать
* вся команда отвечает за качество, а не только тестировщик

**Что такое Scrum?**

Scrum – наиболее популярный Agile-подход, для многих людей согласно статистике эти термины являются синонимами. В целом о Scrum:

* итеративно-инкрементальная разработка. Слово «итеративная» означает, что разработка разбивается на равные по длительности промежутки времени — спринты. Один спринт занимает от одной до четырёх недель. Слово «инкрементальная» подразумевает, что в результате итерации получается новый, потенциально рабочий продукт, решающий бизнес-проблему. Такой продукт называется инкрементом продукта;
* самоорганизующаяся команда, в которой нет проектного менеджера;
* в команде присутствует SCRUM-master;
* в команде есть человек со стороны заказчиков — product owner, или владелец продукта;
* весь период разработки разбит на промежутки времени — спринты. Длина спринта устанавливается в начале проекта и меняется только в том случае, если всплывают неучтённые детали, мешающие уложиться в заданные рамки;
* задачи (функциональные требования, баги, правки заказчика и т.п.) формируют пул работ — бэклог. Изначально в него входят только требования заказчика;
* в начале спринта (и в любом его месте, если это нужно) проводится Backlog Grooming — обработка задач из бэклога. В результате получается проработанный бэклог на 2-3 будущих спринта. Затем PO и SCRUM-команда формулирует цель спринта и ожидаемый результат, и команда составляет бэклог спринта;
* после планирования спринта его состав стараются не менять. Если добавление новых задач всё же происходит, то из спринта исключаются старые и сопоставимые по ценности задачи. Если эти изменения привели к смене цели спринта, то спринт отменяется и планируется заново;
* ежедневные короткие SCRUM-митинги. Они дают понять, как движется процесс, а команда в курсе того, идут ли они к цели спринта или нет;
* в конце спринта выполненные задачи либо подтверждаются, либо отклоняются и возвращаются в бэклог;
* по результатам спринта команда получает инкремент продукта.

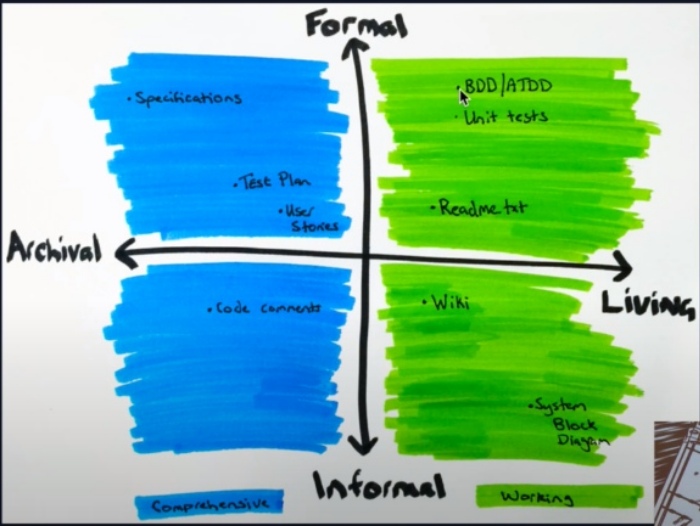
Роли:

|  |  |  |
| --- | --- | --- |
| Product Owner | Scrum Master | Команда |
| определяет особенности продукта. | управляет командой и заботится о продуктивности команды | команда обычно состоит из 5-9 человек. |
| определяет дату релиза и соответствующие функции | поддерживает block список и устраняет барьеры в разработке | в нее входят разработчики, дизайнер, а иногда и тестеры и т. д. |
| устанавливает приоритеты функций в соответствии с рыночной стоимостью и прибыльностью продукта. | координирует все роли и функции | команда организует и планирует свою работу самостоятельно |
| несет ответственность за прибыльность продукта | защищает команду от внешних помех | имеет право делать все в рамках проекта для достижения цели спринта |
| может принять или отклонить результат задания | приглашает на ежедневные разборы, обзор спринта и встречи по планированию | активно участвуют в ежедневных церемониях |

На практике в скрам-тестировании успешность спринта — это когда все задачи, которые были добавлены в Scrum, оказались в статусе Done. Хотя много зависит от опредления, что такое Done в вашем проекте. Скрам по своей сути это просто таймбокс для работ и если не уложились в сроки – то это проблема планирования и на ретроспективе нужно разобраться почему это произошло и зафиксировать на будущее.

**Какие вообще особенности у тестирования в Scrum?**

Классика типа Савина или Канера повествует скорее о традиционных моделях (именно поэтому там уделено столько внимания дизайну, документированию, видам тестирования и т.п., т.к. на всё это там есть время) но в гибких всё значительно отличается. В гибкой методологии времени на разработку тест-кейсов низкого уровня и прочей документации обычно не бывает, поэтому подготавливаются чек-листы. Наборы проверок могут определяться как по не формализованным требованиям, так и на основе рисков (risk based). Ну и тестирование на основе экспертизы – самый простой подход к тестированию, но в то же время и самый рискованный, потому что все тестирование завязывается на экспертизу специалиста, выполняющего тестирование. В некоторых случаях мы всё же можем формализовать Стратегию тестирования (порядок тестирования и подход к выполнению работ по тестированию ПО) и Тест-план (в кратком содержании для спринта не менее 2-х недель, при меньших сроках спринта ведение не актуально). Требования к документации в аджайл: она должна служить потребностям команды, живая документация ценнее архивной.

[](https://camo.githubusercontent.com/77df9ec61321fba325cc94135d7d64da299124052349e04c043ded555e73a68b/68747470733a2f2f73756e392d36352e757365726170692e636f6d2f633230363732342f763230363732343235322f3136653133332f64507866627a2d777266382e6a7067)

3+1 принципа тестирования в аджайл: предотвращение, автоматизация (авто QA - функциональное, приемочное. Devs - юнит, интеграция), гибкость, здравый смысл.  
В agile user stories пишет Product Owner или Business Analyst. Кодеры по ним кодят, а тестеры по ним тестят. ПО или БА к какждой ЮС будут писать критерии приемки, а тестеры на основании их будут писать кейсы. При планировании спринта тестировщик должен выбрать пользовательскую историю из журнала невыполненных работ, которую следует протестировать. Как тестировщик, он / она должен решить, сколько часов (оценка усилий) потребуется, чтобы завершить тестирование для каждой из выбранных пользовательских историй. Как тестер, он / она должен знать, каковы цели спринта. В качестве тестера, внести свой вклад в процесс расстановки приоритетов. Тестер отвечает за разработку сценариев автоматизации. Он планирует автоматизированное тестирование с помощью системы непрерывной интеграции (CI). Автоматизация получает важность из-за коротких сроков доставки.Выполнение нефункционального тестирования для утвержденных пользовательских историй. Тестер взаимодействует с клиентом и владельцем продукта, чтобы определить критерии приемки для приемочных испытаний. В конце спринта тестер также проводит приемочное тестирование (UAT) в некоторых случаях и подтверждает полноту тестирования для текущего спринта. Узнать больше можно по ссылкам в теме "Ты – единственный тестировщик на проекте. Что делать?".

**В чем отличие Kanban от Scrum?**

В отличие от Scrum, в команде канбан отсутствуют роли владельца продукта и модератора, а процесс разработки делится не на универсальные спринты, а на стадии выполнения задач («Планируется», «Разрабатывается», «Тестируется», «Завершено»). Жизненный цикл задачи отображается на канбан-доске, физической или электронной. Такая визуализация делает рабочий процесс открытым и понятным для всех участников, что особенно важно в Agile, когда у команды нет одного формального руководителя.

Канбан, как и другие практики бережливого производства, пришедшие из Японии, направлен на достижение баланса и выравнивание нагрузки исполнителей. Эффективность работы оценивается по среднему времени жизни задачи, от начальной до конечной стадии. Если задача прошла весь путь быстро, то команда проекта работала продуктивно и слаженно. Иначе – необходимо решать проблему: искать, где и почему возникли задержки и чью работу надо оптимизировать

**Что знаете о User stories в гибких подходах к разработке?**

Пользовательские истории (англ. User Story) — способ описания требований к разрабатываемой системе, сформулированных как одно или несколько предложений на повседневном или деловом языке пользователя. Пользовательские истории – это один из самых быстрых способов документирования требований клиента (цель документирования состоит в том, чтобы оперативно и без затрат реагировать на возникающие изменения).

Главное действующее лицо User story – это некий персонаж, который будет совершать какие-либо действия с нашим тестируемым продуктом с учетом его потребностей. Персонаж сопровождается описанием проблем, которые он может (и хочет) решить с помощью нашего продукта. Потребность представляет собой тезис в 1-2 предложения. Для одного пользователя может быть разработано несколько (например, 4-6) User Story.

Для того, чтобы персонажи стали эффективными инструментами проектирования сайта, потребуется не только провести исследование, но и выявить закономерности в поведении пользователей. Как правило, принято создавать и детально прорабатывать одного основного (как показано на mind-map ниже) и несколько второстепенных персонажей.

**Что значит жизненный цикл тестирования ПО? (STLC – Software Testing Lifecycle)**

STLC это модель тестирования, которая предлагает выполнять тестирование систематическим и запланированным способом.

Модель STLC устанавливает следующие этапы:

* Инициация,
* Выявление и анализ требований прямых и косвенных
* Планирование испытаний
* Генерация Test case,
* Отбор показательных Test case,
* Настройка среды
* Проведение проверок,
* Фиксация результатов,
* Анализ результатов,
* Передача информации о соответствии проверенного продукта требованиям.

1. Инициация – событие, которое извещает команду тестирования о необходимости сессии тестирования
2. Выявление требований (RA) – пожалуй, один из главных шагов в процессе тестирования. Необходимо собрать всю доступную информацию о предмете тестирования, вариантах использования и т. п. Первый источник – техническая документация и юзер-стори – это прямые требования. Если некоторые спецификации не являются точными или имеют разногласия, то заинтересованные стороны, такие как бизнес-аналитик (BA), архитекторы, клиенты, дают ясность. Качество же косвенных требований во многом зависят от добросовестности, ответственности, квалификации тестировщика и всей команды проекта.
3. Планирование тестирования (TP) — Как правило, на этом этапе старший QA определяет усилия и смету расходов по проекту, а также готовит и завершает план тестирования. На этом этапе также определяется стратегия тестирования. Команда тестирования выполняет следующие задачи на этапе TP:
   1. Подготовка плана тестирования / стратегии для различных типов тестирования
   2. Выбор тестовых инструментов
   3. Оценка усилий
   4. Планирование ресурсов и определение ролей и обязанностей.
   5. Требования к обучению
   6. Расписание, критерии начала и окончания
   7. Оценка рисков
4. Генерация Test case – выявление всех возможных случаев использования продукта, его характеристик и особенностей в процессе эксплуатации. Это значит: всех случаев, которые тестировщик может «придумать» на основе прямых и косвенных требований, известных ему. Этот этап требует высокой квалификации специалиста по тестированию. При выполнении этой задачи также необходимо подготовить входные данные, необходимые для тестирования. Как только план тестирования будет готов, он должен быть рассмотрен Senior или Lead. Один из документов, который должна подготовить группа, — это матрица отслеживания требований (RTM). Это общеотраслевой стандарт, обеспечивающий правильное сопоставление тест-кейсов с требованиями.
5. Отбор Test case – отбор наиболее показательных, значимых и воспроизводимых Test case. От этого этапа зависит, насколько тестирование будет полезным, эффективным и анализируемым. Количество косвенных требований стремится к бесконечности, и проверять их все подряд – полный абсурд, но подобные кейсы должны быть сгенерированы хотя бы в голове проверяющего.
6. Настройка тестовой среды в STLC — Среда тестирования определяет условия программного и аппаратного обеспечения, при которых тестируется рабочий продукт. Настройка тестовой среды является одним из важнейших аспектов процесса тестирования и может выполняться параллельно с этапом разработки тестового набора. Команда тестирования может быть не вовлечена в это действие, если клиент / команда разработчиков предоставляет среду тестирования, и в этом случае команда тестирования должна выполнить проверку готовности (тестирование дыма) данной среды. Какие задачи выполняются на этапе настройки среды тестирования STLC?
   1. Ознакомление с необходимой архитектурой, настройка среды и подготовка требований к оборудованию и ПО для среды тестирования.
   2. Настройка тестовой среды и тестовых данных
   3. Выполнение Smoke тестирования
7. Проведение проверок – тут все понятно. На этом этапе команда запускает тестовые наборы в соответствии с планом тестирования, определенным в предыдущих шагах, либо adhoc (интуитивно, свободный поиск, без документации). В любом случае это проводится согласно списку отобранных проверок.
8. Фиксация результатов – создание внутренней и внешней тестовой документации в формализованном виде или в виде записей и т. п. На данном этапе отчет о тестировании даже если и создается, то не считается законченным.
9. Анализ результатов – вынесение решения о соответствии проверенного продукта требованиям. Формализация данного решения и его обоснование в виде отчета о тестировании. Сюда также входят процедуры по оценке покрытия требований проверками, тайм-шитинг и пр. Таким образом, проводится анализ не только результатов, но и самой сессии тестирования.
10. Передача информации о соответствии продукта требованиям. Формально: передача внешней тестовой документации заинтересованным в ней сторонам, зачастую инициатору сессии тестирования. В общем случае: помимо документации предоставляется информация о рисках, которые были выявлены в продукте, требованиях, процессах, передаются рекомендации по отработке этих рисков и т. п.

**Что вы знаете о техниках оценки теста? (Test Estimation)**

Оценка теста — это управленческая деятельность, которая приблизительно показывает, сколько времени потребуется для выполнения Задачи. Оценка усилий для теста является одной из основных и важных задач в управлении тестированием (Test Management). Что оценивается?

* Ресурсы необходимы для выполнения любых задач проекта. Это могут быть люди, оборудование, средства, финансирование или что-то еще, что может быть определено для завершения деятельности по проекту. Время — самый ценный ресурс в проекте. Каждый проект имеет срок доставки.
* Человеческие навыки означают знания и опыт членов Команды. Они влияют на вашу оценку. Например, команде, члены которой имеют низкие навыки тестирования, потребуется больше времени для завершения проекта, чем команде, обладающей высокими навыками тестирования.
* Стоимость — это бюджет проекта. Вообще говоря, это означает, сколько денег нужно, чтобы закончить проект.

**В чем разница между SDLC и STLC?**

SDLC определяет все стандартные фазы, которые участвуют в процессе разработки ПО, тогда как процесс STLC определяет различные действия для улучшения качества продукта. SDLC — это жизненный цикл разработки, тогда как STLC - это жизненный цикл тестирования. В SDLC команда разработчиков создает планы проектирования высокого и низкого уровня, в то время как в STLC аналитик тестов создает Систему, План тестирования интеграции. В SDLC разрабатывается реальный код, и фактическая работа выполняется в соответствии с проектной документацией, тогда как в STLC команда тестирования готовит среду тестирования и выполняет тестовые примеры. Жизненный цикл SDLC помогает команде завершить успешную разработку ПО, в то время как фазы STLC охватывают только тестирование ПО.

**Что такое быстрая разработка приложений? (RAD — Rapid Application Development)**

Быстрой разработкой приложений формально является параллельное развитие функций и последующей интеграции. Компоненты/функции развиваются параллельно, как если бы они были мини-проекты, события, time-boxed, доставлены и собраны в работающий прототип. Это может очень быстро дать клиенту что-то увидеть и использовать, и обеспечить обратную связь по поводу доставки и их требований. Быстрое изменение и развитие продукта возможно с помощью этой методики, однако спецификация продукта должна быть разработана для продукта в какой-то момент, и проект должен быть размещен под более формальный контроль перед запуском в производство.

**Что такое разработка через тестирование (TDD — Test Driven Development)?**

В традиционном подходе сначала пишут код, который затем покрывают тестами; в этом подходе сначала разрабатываются тесты, которые в будущем должны быть пройдены кодом. Разработчик будет писать код, пока тесты не начнут проходить. Разработка через тестирование начинается с проектирования и разработки тестов для каждой небольшой функциональности приложения. Также очевидно, в TDD вы получаете 100% покрытия кода тестами.

Есть два уровня TDD:

* Acceptance TDD (ATDD): вы пишете один приемочный тест. Этот тест удовлетворяет требованиям спецификации или удовлетворяет поведению системы. После этого пишете достаточно производственного / функционального кода, чтобы выполнить этот приемочный тест. Приемочный тест фокусируется на общем поведении системы. ATDD также был известен как BDD — Behavioral Driven Development.
* Developer TDD: вы пишете один тест разработчика, то есть модульный тест, а затем просто достаточно производственного кода для выполнения этого теста. Модульное тестирование фокусируется на каждой небольшой функциональности системы. Это называется просто TDD. Основная цель ATDD и TDD — определить подробные, выполнимые требования для вашего решения точно в срок (JIT). JIT означает принятие во внимание только тех требований, которые необходимы в системе, что повышает эффективность.

**TDD в Agile Model Driven Development (AMDD)**

TDD очень хорош в детальной спецификации и валидации. Он не может продумать более важные вопросы, такие как общий дизайн, использование системы или пользовательский интерфейс. AMDD решает проблемы гибкого масштабирования, которых нет в TDD. Таким образом, AMDD используется для больших проблем.

Жизненный цикл AMDD:

* Iteration 0: Envisioning
* Initial requirements envisioning.
* Initial Architectural envisioning.
* Iteration modeling.
* Model storming.
* Test Driven Development (TDD).
* Reviews.

TDD сокращает цикл обратной связи программирования, AMDD сокращает цикл обратной связи моделирования. TDD — детальная спецификация, AMDD работает для больших проблем. TDD способствует разработке качественного кода, AMDD способствует качественному общению с заинтересованными сторонами и разработчиками. TDD общается с программистами, AMDD общается с бизнес-аналитиками, заинтересованными сторонами и специалистами по данным. TDD не визуально ориентированный, AMDD визуально ориентированный. TDD имеет ограниченную область применения, AMDD имеет широкий охват. Это предполагает работу по достижению общего понимания. Оба поддерживают эволюционное развитие.

**Тестирование на основе моделей (MDD — Model-driven Development)**

Это метод тестирования черного ящика, при котором поведение тестируемого программного обеспечения во время выполнения проверяется на основе прогнозов, сделанных моделями. Модель — это описание поведения системы. Поведение может быть описано в виде наглядной схемы, Data Flow, Control Flow, Dependency Graphs, Decision Tables, State transition machines или mind map. Простой аналогией модели в тестировании является электрическая схема при разработке электроприбора. Этот подход к тестированию требуется, когда высока цена ошибки в большом продукте и нужно как можно раньше попытаться ее предотвратить.

**Тестирование на основе данных (DDT — Data Driven testing)**

DATA DRIVEN testing — это инфраструктура автоматизации тестирования, которая хранит тестовые данные в виде таблицы. Это позволяет инженерам по автоматизации иметь единый сценарий тестирования, который может выполнять тесты для всех тестовых данных в таблице. В этой структуре входные значения считываются из файлов данных и сохраняются в переменную в тестовых сценариях. Ddt (тестирование на основе данных) позволяет объединять как положительные, так и отрицательные Test case в один тест. В среде автоматизации тестирования на основе данных входные данные могут храниться в одном или нескольких источниках данных, таких как xls, XML, csv и базы данных.

Часто у нас есть несколько наборов данных, на которых мы должны запускать одни и те же тесты. Создание отдельного теста для каждого набора данных — длительный и трудоемкий процесс. Data Driven testing Framework решает эту проблему, отделяя данные от функциональных тестов. Один и тот же сценарий тестирования может выполняться для различных комбинаций входных данных теста и генерировать результаты теста. Например, мы хотим протестировать вход в систему с несколькими полями ввода с 1000 различными наборами данных. Чтобы проверить это, вы можете использовать следующие разные подходы: Подход 1) Создайте 1000 сценариев по одному для каждого набора данных и запускайте каждый тест отдельно по одному. Подход 2) Вручную измените значение в тестовом скрипте и запустите его несколько раз. Подход 3) Импортируйте данные из листа Excel. Извлеките данные теста из строк Excel по очереди и выполните скрипт. В приведенных трех сценариях первые два являются слишком трудоемкими. Таким образом, идеально следовать третьему подходу и это не что иное, как Data-Driven Framework.

**Тестирование на основе риска (RBT — Risk Based Testing)**

ТЕСТИРОВАНИЕ НА ОСНОВЕ РИСКА (RBT) — это тип тестирования, основанный на вероятности риска. Он включает в себя оценку риска на основе сложности, критичности бизнеса, частоты использования, видимых областей, областей, подверженных дефектам, и т. д. Он включает определение приоритетов тестирования модулей и функций тестируемого приложения на основе влияния и вероятности отказов.

Риск — это возникновение неопределенного события, которое положительно или отрицательно влияет на измеримые критерии успеха проекта. Это могут быть события, которые произошли в прошлом или текущие события, или что-то, что может произойти в будущем. Эти неопределенные события могут повлиять на стоимость, бизнес, технические и качественные цели проекта. Позитивные риски упоминаются как возможности и помощь в устойчивости бизнеса. Например, инвестирование в новый проект, изменение бизнес-процессов, разработка новых продуктов. Отрицательные риски называются угрозами, и для успеха проекта должны быть реализованы рекомендации по их минимизации или устранению.

Примерный чеклист:

* Важные функциональные возможности в проекте.
* Видимая пользователю функциональность в проекте
* Функциональность, оказывающая наибольшее влияние на безопасность
* Функциональные возможности, которые оказывают наибольшее финансовое влияние на пользователей
* Сложные области исходного кода и кодов, подверженных ошибкам
* Функции, которые могут быть проверены в начале цикла разработки.
* Особенности или функциональные возможности, которые были добавлены в дизайн продукта в последнюю минуту.
* Критические факторы подобных / связанных предыдущих проектов, которые вызвали проблемы.
* Основные факторы или проблемы аналогичных / связанных проектов, которые оказали огромное влияние на эксплуатационные расходы.
* Плохие требования, которые приводят к плохим проектам и тестам, которые могут повлиять на цели и результаты проекта.
* В худшем случае продукт может быть настолько дефектным, что его невозможно переработать, и его необходимо полностью утилизировать, что может нанести серьезный ущерб репутации компании. Определите, какие проблемы имеют решающее значение для целей продукта.
* Ситуации или проблемы, которые могут привести к постоянным жалобам на обслуживание клиентов. Сквозные тесты могут легко сфокусироваться на нескольких функциональных возможностях системы. Оптимальный набор тестов, которые могут максимизировать покрытие риска
* Какие тесты будут иметь лучшее соотношение риска и требуемого времени?

**Что вы знаете о потоковом тестировании? (BFT — BusinessFlowTesting)**

BFT – это подход к тестированию, где вы смотрите на продукт не с точки зрения конкретных кейсов, а с точки зрения поведения системы в каждой ее точке. Подход является объединением таких концепций, как Data Driven testing и Behaviour Driven testing, примененным к бизнес-моделям, хорошо описываемым графами движения данных. Вы не тестируете отдельные тест-кейсы, а проверяете работоспособность системы, тест-кейсы получаются сами по себе. Образно говоря, у вас есть:

* Набор разнообразных входных бизнес данных
* Система, разбитая на важные для тестирования состояния бизнес-сущностей, которые надо проверить
* Правила перехода между состояниями

У вас нет тест-кейсов. Тест-кейсы генерируются на основании входных данных сами. Больше не надо спорить, как назвать тест-кейсы правильно. Название тест-кейса – это набор определяющих его входных данных и путь, по которым они пройдут. Название получается длинное, но полностью описывающее данный тест. И к тому же вы не тратите на него ни секунды — оно создается само.

**Тестирование в разных сферах/областях (testing different domains)**

**Что такое веб-тестирование и как его производить?**

ВЕБ-ТЕСТИРОВАНИЕ, или тестирование веб-сайта, проверяет ваше веб-приложение или веб-сайт на наличие потенциальных ошибок, прежде чем оно будет опубликовано и доступно для широкой публики.

1. Функциональное тестирование: используется для проверки того, соответствует ли ваш продукт спецификациям, а также функциональным требованиям, которые вы наметили для него в документации по разработке. Включает в себя:

* Проверка, что все ссылки на ваших веб-страницах работают правильно и что нет битых ссылок.
  + Исходящие ссылки
  + Внутренние ссылки
  + Якорные ссылки (слово или словосочетание, на котором поставлена ссылка)
  + MailTo Ссылки
* Текстовые формы работают как положено.
  + Проверка скриптов в форме работает как положено. Например, если пользователь не заполняет обязательное поле в форме, отображается сообщение об ошибке.
  + Проверьте значения по умолчанию
  + После отправки данные в формах отправляются в базу данных или связываются с рабочим адресом электронной почты.
  + Формы оптимально отформатированы для лучшей читаемости
* Тестовые куки работают как положено. Файлы cookie — это небольшие файлы, используемые веб-сайтами для запоминания активных пользовательских сессий, поэтому вам не нужно входить в систему каждый раз, когда вы посещаете веб-сайт. Тестирование файлов cookie будет включать
  + Тестовые файлы cookie (сеансы) удаляются либо после очистки кэша, либо по истечении срока их действия. Удалите файлы cookie (сеансы) и проверьте, запрашиваются ли учетные данные при следующем посещении сайта.
  + Протестируйте HTML и CSS, чтобы поисковые системы могли легко сканировать ваш сайт. Это будет включать
    - Проверка на синтаксические ошибки
    - Удобочитаемые цветовые схемы
    - Стандартное соответствие. Убедитесь, что соблюдаются такие стандарты, как W3C, OASIS, IETF, ISO, ECMA или WS-I.
* Тест бизнес-воркфлоу — это будет включать в себя
  + Тестирование вашего end-to-end workflow / бизнес-сценариев
  + Также проверьте отрицательные сценарии, чтобы при выполнении пользователем неожиданного шага в веб-приложении отображалось соответствующее сообщение об ошибке или справка.
* Примеры функциональных тест-кейсов:
  + Все обязательные поля должны быть валидированы.
  + Звездочка должна отображаться для всех обязательных полей.
  + Не должно отображаться сообщение об ошибке для дополнительных полей.
  + Проверьте, что високосные годы проверены правильно и не вызывают ошибок.
  + Числовые поля не должны принимать буквы и должно отображаться соответствующее сообщение об ошибке.
  + Проверьте наличие отрицательных чисел, если это разрешено для числовых полей.
  + Тестовое деление на ноль должно быть правильно обработано.
  + Проверьте максимальную длину каждого поля, чтобы убедиться, что данные не усекаются.
  + Тест всплывающего сообщения («Это поле ограничено 500 символами») должно отображаться, если данные достигают максимального размера поля.
  + Проверьте, должно ли отображаться подтверждающее сообщение для операций обновления и удаления.
  + Величины должны быть в подходящем формате.
  + Проверьте все поля ввода на ввод специальных символов.
  + Проверьте функциональность тайм-аута.
  + Проверьте функциональность сортировок.
  + Проверьте, что FAQ и Политика конфиденциальности четко определены и доступны для пользователей.
  + Проверьте, всё ли работает и не перенаправляется ли пользователь на страницу ошибки.
  + Все загруженные документы открываются правильно.
  + Пользователь должен иметь возможность скачать загруженные файлы.
  + Проверьте функциональность электронной почты системы. Тестируемый скрипт корректно работает в разных браузерах (IE, Firefox, Chrome, Safari и Opera).
  + Проверьте, что произойдет, если пользователь удалит файлы cookie, находясь на сайте.
  + Проверьте, что произойдет, если пользователь удалит файлы cookie после посещения сайта.

2. Юзабилити-тестирование стало важной частью любого веб-проекта. Его могут провести тестировщики или небольшая фокус-группа, похожая на целевую аудиторию веб-приложения.

* Навигация:
  + Меню, кнопки или ссылки на разные страницы вашего сайта должны быть легко видны и согласованы на всех веб-страницах.
* Проверьте содержимое:
  + Содержание должно быть разборчивым, без орфографических или грамматических ошибок.
  + Изображения, если они присутствуют, должны содержать «альтернативный» текст
* Примеры тестов юзабилити:
  + Содержание веб-страницы должно быть правильным без каких-либо орфографических или грамматических ошибок
  + Все шрифты должны быть в соответствии с требованиями.
  + Весь текст должен быть правильно выровнен.
  + Все сообщения об ошибках должны быть правильными без каких-либо орфографических или грамматических ошибок, а сообщение об ошибке должно соответствовать метке поля.
  + Текст подсказки должен быть там для каждого поля.
  + Все поля должны быть правильно выровнены.
  + Должно быть достаточно места между метками полей, столбцами, строками и сообщениями об ошибках.
  + Все кнопки должны быть в стандартном формате и размере.
  + Домашняя ссылка должна быть на каждой странице.
  + Отключенные поля должны быть недоступны.
  + Проверьте наличие битых ссылок и изображений.
  + Сообщение о подтверждении должно отображаться для любого вида операции обновления и удаления. Проверить сайт на разных разрешениях (640 х 480, 600х800 и т. д. )
  + Убедитесь, что вкладка должна работать правильно.
  + Полоса прокрутки должна появляться только при необходимости.
  + Если при отправке появляется сообщение об ошибке, информация, заполненная пользователем, должна быть там.
  + Название должно отображаться на каждой веб-странице
  + Все поля (текстовое поле, раскрывающийся список, переключатель и т. д. ) И кнопки должны быть доступны с помощью сочетаний клавиш, и пользователь должен иметь возможность выполнять все операции с помощью клавиатуры.
  + Проверьте, не усекаются ли выпадающие данные из-за размера поля.
  + Также проверьте, жестко ли закодированы или управляются данные через администратора.

3. Тестирование интерфейсов: Здесь тестируются три области: приложение, веб-сервер и сервер базы данных.

* Приложение: тестовые запросы правильно отправляются в базу данных и вывод на стороне клиента отображается правильно. Ошибки, если таковые имеются, должны быть обнаружены приложением и должны отображаться только администратору, а не конечному пользователю.
* Веб-сервер: тестовый веб-сервер обрабатывает все запросы приложений без какого-либо отказа в обслуживании.
* Сервер базы данных: убедитесь, что запросы, отправленные в базу данных, дают ожидаемые результаты. Проверьте реакцию системы, когда невозможно установить соединение между тремя уровнями (Приложение, Интернет и База данных) и соответствующее сообщение отображается конечному пользователю.

4. Тестирование базы данных: База данных является одним из важнейших компонентов вашего веб-приложения, и необходимо тщательно провести тестирование. Тестирование будет включать в себя:

* Проверьте, отображаются ли какие-либо ошибки при выполнении запросов
* Целостность данных поддерживается при создании, обновлении или удалении данных в базе данных.
* Проверьте время ответа на запросы.
* Тестовые данные, полученные из вашей базы данных, точно отображаются в вашем веб-приложении.
* Примеры тест-кейсов для тестирования базы данных:
  + Проверьте имя базы данных: имя базы данных должно соответствовать спецификациям.
  + Проверьте таблицы, столбцы, типы столбцов и значения по умолчанию: все должно соответствовать спецификациям.
  + Проверьте, допускает ли столбец null значение или нет.
  + Проверьте первичный и внешний ключ каждой таблицы.
  + Проверьте хранимую процедуру:
  + Проверьте, установлена ли сохраненная процедура или нет.
  + Проверьте имя хранимой процедуры
  + Проверьте имена параметров, типы и количество параметров.
  + Проверьте требуемые параметры.
  + Проверьте хранимую процедуру, удалив некоторые параметры
  + Проверьте, когда выход равен нулю, это должно повлиять на нулевые записи.
  + Проверьте хранимую процедуру, написав простые запросы SQL.
  + Проверьте, возвращает ли хранимая процедура значения
  + Проверьте хранимую процедуру с образцами входных данных.
  + Проверьте поведение каждого флага в таблице.
  + Убедитесь, что данные правильно сохраняются в базе данных после каждой отправки страницы.
  + Проверьте данные, если выполняются операции DML (Обновить, удалить и вставить).
  + Проверьте длину каждого поля: длина поля на Frontend и backend должна быть одинаковой.
  + Проверьте имена баз данных QA, UAT и production. Имена должны быть уникальными.
  + Проверьте зашифрованные данные в базе данных.
  + Проверьте размер базы данных.
  + Также проверьте время ответа каждого выполненного запроса.
  + Проверьте данные, отображаемые на Frontend, и убедитесь, что они совпадают с backend.
  + Проверьте достоверность данных, вставив неверные данные в базу данных.
  + Проверьте триггеры.

5. Тестирование на совместимость. Тесты на совместимость гарантируют, что ваше веб-приложение правильно отображается на разных устройствах.

* Вам нужно проверить, правильно ли отображается ваше веб-приложение в браузерах, работает ли JavaScript, AJAX и аутентификация нормально. Вы также можете проверить совместимость мобильного браузера. Рендеринг веб-элементов, таких как кнопки, текстовые поля и т. д. , изменяется с изменением в операционной системе. Убедитесь, что ваш сайт работает нормально для различных комбинаций операционных систем, таких как Windows, Linux, Mac и браузеров, таких как Firefox, Internet Explorer, Safari и т. д.
* Примеры тестов на совместимость:
  + Протестируйте сайт в разных браузерах (IE, Firefox, Chrome, Safari и Opera) и убедитесь, что сайт отображается правильно.
  + Используемая версия HTML совместима с соответствующими версиями браузера.
  + Проверьте правильность отображения изображений в разных браузерах.
  + Протестируйте шрифты, которые можно использовать в разных браузерах.
  + Протестируйте код Javascript в разных браузерах.
  + Проверьте анимированные GIF-файлы в разных браузерах.

6. Тестирование производительности: Это нужно, чтобы обеспечить работу вашего сайта при любых нагрузках. Деятельность по тестированию ПО будет включать, но не ограничиваться:

* Время отклика приложения сайта на разных скоростях соединения
* Нагрузочное тестирование вашего веб-приложения, чтобы определить его поведение при нормальной и пиковой нагрузке.
* Стресс-тест вашего веб-сайта, чтобы определить его точку остановки при превышении нормальных нагрузок в пиковое время.
* Проверьте, происходит ли сбой из-за пиковой нагрузки, как сайт восстанавливается после такого события, убедитесь, что методы оптимизации, такие как сжатие gzip и кэш включены, чтобы сократить время загрузки

7. Тестирование безопасности жизненно важно для сайта электронной коммерции, который хранит конфиденциальную информацию о клиентах, например, кредитные карты. Деятельность по тестированию будет включать:

* Проверка несанкционированного доступа к защищенным страницам
* Запрещенные файлы не должны быть загружаемыми без соответствующего доступа
* Сессии автоматически прекращаются после длительного отсутствия активности пользователя
* При использовании SSL-сертификатов веб-сайт должен перенаправить на зашифрованные SSL-страницы.
* Примеры тестовых сценариев для тестирования безопасности:
  + Убедитесь, что веб-страница, содержащая важные данные, такие как пароль, номера кредитных карт, секретные ответы на секретный вопрос и т. д. , Должна быть отправлена через HTTPS (SSL).
  + Убедитесь, что важная информация, такая как пароль, номера кредитных карт и т. д. , Должна отображаться в зашифрованном виде.
  + Правила проверки пароля применяются на всех страницах аутентификации, таких как Регистрация, забытый пароль, смена пароля.
  + Убедитесь, что, если пароль изменен, пользователь не должен иметь возможность войти со старым паролем. Убедитесь, что сообщения об ошибках не должны отображать важную информацию.
  + Убедитесь, что, если пользователь вышел из системы или сеанс пользователя истек, пользователь не должен перемещаться по сайту авторизованным.
  + Проверьте доступ к защищенным и незащищенным веб-страницам напрямую без входа в систему.
  + Убедитесь, что опция «Просмотр исходного кода» отключена и не должна быть видна пользователю. Убедитесь, что учетная запись пользователя заблокирована, если пользователь вводит неправильный пароль несколько раз.
  + Убедитесь, что куки не должны хранить пароли.
  + Убедитесь, что, если какая-либо функция не работает, система не должна отображать информацию о приложении, сервере или базе данных. Вместо этого она должна отображать пользовательскую страницу ошибки.
  + Проверьте атаки SQL-инъекций.
  + Проверьте роли пользователей и их права. Например, запрашивающая сторона не должна иметь доступа к странице администратора.
  + Убедитесь, что важные операции записаны в файлы журналов, и эта информация должна быть отслеживаемой.
  + Убедитесь, что значения сеанса находятся в зашифрованном формате в адресной строке.
  + Убедитесь, что информация о файлах cookie хранится в зашифрованном формате.
  + Проверьте приложение на брутфорс-атаки

8. Тестирование толпы (Crowdtesting): Вы берете большое количество людей (толпу) для выполнения тестов, которые в противном случае были бы выполнены выбранной группой людей в компании. Краудсорсинговое тестирование представляет собой интересную и перспективную концепцию и помогает выявить многие незамеченные дефекты. Оно включает в себя практически все типы тестирования, применимые к вашему веб-приложению.

**Тестирование банковского ПО**

Сектор BFSI (банковские, финансовые услуги и страхование) является крупнейшим потребителем ИТ-услуг. Банковские приложения напрямую связаны с конфиденциальными финансовыми данными. Обязательно, чтобы все операции, выполняемые банковским программным обеспечением, выполнялись без сбоев и ошибок. Банковское ПО выполняет различные функции, такие как перевод и внесение средств, запрос баланса, история транзакций, вывод средств и так далее. Тестирование банковского приложения гарантирует, что эти действия не только хорошо выполняются, но и остаются защищенными от хакеров.

Важно отметить стандартные функции, ожидаемые от любого банковского приложения:

* Оно должно поддерживать тысячи одновременных пользовательских сессий
* Банковское приложение должно интегрироваться с другими многочисленными приложениями, такими как торговые счета, утилита оплаты счетов, кредитные карты и т. д.
* Должно обрабатывать быстрые и безопасные транзакции
* Оно должно включать в себя массивную систему хранения.
* Для устранения проблем с клиентами у него должна быть высокая возможность аудита
* Оно должен обрабатывать сложные бизнес-процессы
* Нужно поддерживать пользователей на разных платформах (Mac, Linux, Unix, Windows)
* Оно должно поддерживать пользователей из разных мест и на разных языках
* Поддерживать пользователей в различных платежных системах (VISA, AMEX, MasterCard)
* Оно должно поддерживать несколько секторов обслуживания (кредиты, розничные банковские операции и т. д.)
* Иметь механизм защиты от катастрофических сбоев

**Тестирование электронной коммерции (eCommerce)**

Тестирование электронной коммерции помогает в предотвращении ошибок и повышает ценность продукта, обеспечивая соответствие требованиям клиента. Целями тестирования являются:

* Обеспечение надежности и качества ПО
* Уверенность в системе
* Оптимальная производительность

Настройка системы электронной коммерции является сложным процессом и зависит от множества рыночных переменных. Для поддержания целостности системы электронной коммерции тестирование становится обязательным. Что проверяется:

* Совместимость браузера:
  + Поддержка для старых браузеров
  + Специальные браузерные расширения
  + Тестирование браузера должно охватывать основные платформы (Linux, Windows, Mac и т. д. )
* Отображение страниц:
  + Некорректное отображение страниц
  + Сообщения об ошибках во время выполнения
  + Плохое время загрузки страницы
  + Битые ссылки, зависимость от плагина, размер шрифта и т. д.
* Управление сессиями
  + Истечение сессии
  + Хранение сессии
* Удобство и простота
  + Неинтуитивный дизайн
  + Плохая навигация по сайту
  + Навигация по каталогам
  + Отсутствие помощи-поддержки
* Анализ содержимого
  + Вводящий в заблуждение, оскорбительный или незаконный контент
  + Роялти-фри изображения и нарушение авторских прав
  + Функциональность персонализации
  + Доступность 24/7
* Доступность
  + Атаки отказа в обслуживании
  + Неприемлемые уровни недоступности
* Резервное копирование и восстановление
  + Сбой или отказ восстановления
  + Ошибка резервного копирования
  + Отказоустойчивость
* Операции
  + Целостность транзакции
  + Пропускная способность
  + Аудит
* Обработка заказов на покупку и покупка
  + Функциональность корзины
  + Обработка заказов
  + Процесс оплаты
  + Отслеживание заказа
* Интернационализация
  + Языковая поддержка
  + Отображение языков
  + Культурная чувствительность
  + Региональный учет
* Оперативные бизнес-процедуры
  + Насколько хорошо справляется электронная процедура
  + Наблюдение за узкими местами
* Системная интеграция
  + Формат интерфейса данных
  + Обновления
  + Величина нагрузки интерфейса
  + Интегрированная производительность
* Производительность
  + Узкие места производительности
  + Обработка нагрузки
  + Анализ масштабируемости
* Логин и безопасность
  + Возможность входа
  + Проникновение и контроль доступа
  + Небезопасная передача информации
  + Веб-атаки
  + Компьютерные вирусы
  + Цифровые подписи

Тестирование производительности — главный приоритет в электронной коммерции. Просто задержите около 250 миллисекунд времени загрузки страницы – и это заставляет вашего клиента идти к вашему конкуренту. Гигант розничной торговли Walmart пересмотрел скорость своего сайта и заметил увеличение на 2% коэффициента конверсии посетителей и доходов на 1%. Эффективность вашего сайта зависит от этих факторов:

* Пропускная способность (Throughput):
  + Запросов в секунду
  + Транзакций в минуту
  + Выполнений за клик
* Время отклика (Response Time):
  + Длительность задачи
  + Секунд на клик
  + Загрузка страницы
  + DNS Lookup
  + Продолжительность времени между кликом и просмотром страницы

**Тестирование платежного шлюза (Payment Gateway)**

Платежный шлюз — это сервис приложений электронной коммерции, который принимает оплату кредитной картой для покупок в Интернете. Платежные шлюзы защищают данные кредитной карты, шифруя конфиденциальную информацию, такую как номера кредитных карт, данные владельца счета и так далее. Эта информация безопасно передается между покупателем и продавцом, и наоборот. Современные платежные шлюзы также надежно подтверждают платежи с помощью дебетовых карт, электронных банковских переводов, банковских карт, бонусных баллов и т. д.

Типы платежных систем:

* Собственный местный платежный шлюз: Хостинговая система шлюзов платежей направляет клиента от сайта электронной коммерции к шлюзу во время процесса оплаты. Как только платеж будет выполнен, он вернет клиента на сайт электронной коммерции. Для такого типа оплаты вам не нужен идентификатор продавца, например, хостинговый платежный шлюз — PayPal, Noche и WorldPay.
* Shared (встраиваемый у партнёров) платежный шлюз: В разделяемом платежном шлюзе при обработке платежа клиент направляется на страницу оплаты и остается на сайте электронной коммерции. Как только реквизиты платежа заполнены, процесс оплаты продолжается. Поскольку он не покидает сайт электронной коммерции во время обработки платежа, этот режим прост и, более предпочтителен, примером шлюза с общими платежами является eWay, Stripe.

Тестирование для Платежного шлюза должно включать:

* Функциональное тестирование: это тестирование базовой функциональности платежного шлюза. Оно предназначено для проверки того, ведет ли себя приложение ожидаемым образом при обработке заказов, расчетах, добавлении НДС в зависимости от страны и т. д.
* Интеграция: Проверьте интеграцию с вашей кредитной картой.
* Производительность. Определите различные показатели производительности, такие как максимально возможное количество пользователей, проходящих через шлюзы в течение определенного дня, и конвертирующих их в одновременных пользователей.
* Безопасность: вам необходимо выполнить глубокую проверку безопасности для Платежного шлюза

Примеры тест-кейсов для тестирования платежного шлюза:

* В процессе оплаты попробуйте изменить язык платежного шлюза.
* После успешной оплаты проверьте все необходимые компоненты
* Проверьте, что произойдет, если платежный шлюз перестанет отвечать во время оплаты
* В процессе оплаты проверьте, что произойдет, если сессия заканчивается
* В процессе оплаты проверьте, что происходит в бэкэнде
* Проверьте, что произойдет, если процесс оплаты не удастся
* Проверьте записи базы данных, хранят ли они данные кредитной карты или нет
* В процессе оплаты проверяйте страницы ошибок и страницы безопасности
* Проверьте при наличии блокировщика всплывающих окон
* Между платежным шлюзом и страницами проверьте буферные страницы.
* Проверка успешной оплаты, код успеха отправляется в приложение и пользователю отображается страница подтверждения
* Убедитесь, что транзакция обрабатывается немедленно или обработка передана вашему банку.
* После успешной транзакции проверьте, возвращается ли платежный шлюз в ваше приложение.
* Проверьте все форматы и сообщения при успешном процессе оплаты
* Если у вас нет квитанции об авторизации от платежного шлюза, товар не должен быть отправлен
* Сообщите владельцу о любой транзакции, обработанной по электронной почте. Шифровать содержимое почты.
* Проверьте формат суммы с форматом валюты
* Проверьте, доступен ли каждый из вариантов оплаты
* Проверьте, открывает ли каждый перечисленный способ оплаты соответствующий способ оплаты в соответствии со спецификацией.
* Убедитесь, что в платежном шлюзе по умолчанию выбран нужный вариант дебетовой / кредитной карты.
* Проверьте опцию по умолчанию для дебетовых карт — показывает выпадающее меню выбора карты

**Тестирование систем розничной торговли (POS — Point Of Sale)**

POS-тестирование определяется как тестирование приложения в точках продаж. ПО POS или Point Of Sale — это жизненно важное решение для предприятий розничной торговли, позволяющее легко совершать розничные транзакции из любого места. Вы, наверное, видели терминал торговой точки в своем любимом торговом центре. Система является более сложной, чем вы думаете, и тесно интегрирована с другими программными системами, такими как Склад, Инвентарь, Заказ на поставку, Цепочка поставок, Маркетинг, Планирование товаров и т. д. Знание предметной области POS важно для тестирования.

Оценка POS-системы может быть разбита на два уровня:

* Уровень применения (Application Level)
* Уровень предприятия (Enterprise Level)

|  |  |
| --- | --- |
| Сценарий | Кейсы |
| Деятельность кассира | * Проверьте правильность ввода товаров, приобретенных клиентом * Тестовые скидки применяются правильно * Убедитесь, что платежные карты магазина (value cards) могут быть использованы * Управление мелочью работает как положено * Проверьте соответствие итогов и закрытия * Убедитесь, что денежный ящик кассы работает правильно * Проверьте, что система POS совместима с периферийными устройствами, такими как считыватель RFID, сканер штрих-кода и т. д. |
| Обработка платежного шлюза | * Проверка правильности CVV кредитных карт * Тест на использование карт с обеих сторон * Убедитесь, что данные карты правильно зашифрованы и расшифрованы |
| Продажи | * Проверьте для обычного процесса продажи * Продажи могут быть обработаны дебетовой / кредитной картой * Проверить покупку по карте лояльности * Проверка правильности отображаемой цены для покупаемых товаров * Тест для «0» или нулевой транзакции * Привязка UPC или штрих-кодов с поставщиками * Проверка платежных данных или данных о доставке в диспетчере платежей * Тест для reference транзакции * Проверьте формат печати сгенерированной квитанции * Убедитесь, что для утвержденных, удержанных или отклоненных транзакций создан правильный код |
| Сценарии возврата и обмена | * Убедитесь, что внутренняя опись хорошо интегрирована с другими торговыми точками или цепочкой поставок * Чек на обмен или возврат товара наличными * Проверьте, работает ли система при обмене или возврате товара с помощью кредитной карты * Проверка системы обработки продажи с чеком или без чека * Убедитесь, что система должна позволять вводить штрих-код вручную, если сканер не работает * Убедитесь, что система отображает как текущую сумму, так и сумму скидки при обмене товара, если применимо |
| Производительность | * Проверьте скорость или время, необходимое для получения ответа или отправки запроса * Проверьте, применяются ли правила транзакций (скидки / налоги и т. д.) * Убедитесь, что для утвержденных, удержанных или отклоненных транзакций создан правильный код |
| Негативные сценарии | * Тестовая система с просроченными данными карты * Тест с неверным PIN-кодом для кредитной карты * Проверьте инвентарь, введя неправильный код товара * Проверьте, как система реагирует при вводе неверного номера счета * Тест на отрицательную транзакцию * Проверьте ответ системы при вводе недопустимой даты для рекламных предложений онлайн-товаров |
| Управление акциями и скидками | * Проверка: система для различных скидок, таких как ветеранская скидка, сезонная скидка, скидка на покупку или перелет и т. д. * Проверка: система для различных рекламных предложений по отдельным позициям * Проверка: система оповещений, которая уведомляет об окончании или начале сезонных предложений * Проверьте, распечатывает ли квитанция точную скидку или предложения, которые используются * Проверка: система для определения неправильных предложений или скидок на товары онлайн * Протестируйте процесс управления заказами * Убедитесь, что данные продукта, полученные после сканирования штрих-кода, являются точными |
| Отслеживание данных клиента | * Проверка ответа системы с неверным вводом данных клиента * Проверка: система для разрешения авторизованного доступа к конфиденциальным данным клиента * Протестируйте базу данных для записи истории покупок клиента (что он покупает, как часто он покупает и т. д. ) |
| Безопасность и соответствие нормативным требованиям | * Проверка системы POS на соответствие нормативным требованиям * Проверка: система оповещения -> security defenders * Убедитесь, что вы можете аннулировать платеж перед отправкой * Тестируйте профили пользователей и уровни доступа в POS Software * Проверка согласованности базы данных * Проверьте конкретную информацию о каждой наличности, идентификатор купона, номер чека и т. д. |
| Тестирование отчетности | * Тестирование отчета по анализу трендов * Тестовая информация, связанная с транзакцией по кредитной карте, должна быть отражена в отчетах * Тест для отдельных, а также сводные отчеты истории покупок клиентов * Тест для генерации онлайн отчетов |

**Тестирование в сфере страхования (Insurance)**

Страховые компании в значительной степени полагаются на ПО для ведения своего бизнеса. Программные системы помогают им заниматься различными видами страховой деятельности, такими как разработка стандартных форм полисов, обработка процесса выставления счетов, управление данными клиента, оказание качественных услуг клиенту, координация между филиалами и так далее. Хотя это ПО разработано с учетом ожиданий заказчика, его надежность и согласованность должны быть проверены перед его фактическим внедрением. Тестирование ПО гарантирует качество страхового ПО, выявляя ошибки перед запуском.

Страхование определяется как справедливая передача риска убытков от одного субъекта другому в обмен на платеж. Страховая компания, которая продает полис, называется СТРАХОВОЙ, а лицо или компания, которая использует полис, называется ЗАСТРАХОВАННЫЙ. Страховые полисы обычно делятся на две категории, и страховщик покупает эти полисы в соответствии с их требованиями и бюджетом. Тем не менее, есть другие виды страхования, которые подпадают под эти категории:

* Страхование по безработице (Unemployment insurance)
* Социальное обеспечение (Social Security)
* Компенсация рабочим (Workers Compensation)

Есть много ветвей в страховой компании, которые требуют тестирования:

* Системы администрирования политики (Policy Administration Systems)
* Системы управления претензиями (Claim Management Systems)
* Системы управления распределением (Distribution Management Systems)
* Системы управления инвестициями (Investment Management Systems)
* Сторонние системы администрирования (Third party Administration Systems)
* Решения по управлению рисками (Risk Management Solutions)
* Регулирование и соответствие (Regulatory and Compliance)
* Актуарные системы (Оценка и ценообразование) (Actuarial Systems (Valuation & Pricing))

Сектор страхования представляет собой сеть небольших подразделений, которая прямо или косвенно занимается обработкой требований. Для бесперебойного функционирования страховой компании необходимо, чтобы каждое из этих подразделений было тщательно проверено для достижения желаемого результата. Тестирование включает в себя:

* Колл-центр (Call Center):
  + Интеграционное тестирование IVR (IVR [Integration testing](https://www.guru99.com/integration-testing.html))
  + Маршрутизация и назначение вызовов (Call routing and assignment)
  + Безопасность и доступ (Security and access)
  + Рефлексивные вопросы (Reflexive Questions)
* Политика обслуживания (Policy Serving):
  + Тестирование жизненного цикла политики (Policy life cycle testing)
  + Изменения в финансовой и нефинансовой политике (Financial and Non-financial policy changes)
  + Политика недействительности и восстановления (Policy lapse and Re-instatement)
  + Оповещения о страховых выплатах (Premium due alerts)
  + Оценка NPV/NAV (Valuation of NPV/NAV)
* Претензии (Claims):
  + Сортировка и уступка требований (Claims triage and assignment)
  + Тестирование жизненного цикла претензий (Testing claims life cycle)
  + Учет требований / резервирование (Claims accounting/reserving)
  + EDI / обмен сообщениями от третьих лиц (Third party EDI/messaging)
* Прямой канал (Direct channel):
  + Мобильный доступ (Mobile access)
  + Кросс-браузерность / кроссплатформенность (Cross browser/cross platform accessibility)
  + Производительность приложения (Application performance)
  + Удобство использования приложения (Usability of application)
* Отчеты / BI (Reports/BI):
  + Соблюдение нормативных требований (Behaving to regulatory requirements)
  + Генерация качественных данных для отчетности (Generate quality data for reporting)
  + Создание массовых данных для сводных отчетов (Create bulk data for roll-up reports)
  + Тестирование полей на основе формул в отчетах (Testing formula based fields in reports)
* Андеррайтинг (Underwriting):
  + Качество андеррайтинга (Underwriting quality)
  + Ручная и прямая обработка (Manual and Straight through processing)
  + Сложные бизнес-правила (Complex business rules)
  + Рейтинг эффективности (Rating efficiency)
  + Управление требованиями (Vendor Interfacing) (Requirements Management)
* Интеграция (Integration):
  + Интеграция данных (Data integration)
  + Комплексная интеграция интерфейса (Complex interface integration)
  + Форматы источника / назначения (Source/Destination formats)
  + Производственный интерфейс (Production like interface)
  + Эффективность пулла/пуша веб-сервиса (Web service pull/push efficiency)
* Новый бизнес (New Business):
  + Проверить комбинации коэффициентов (Validate rates-factor combinations)
  + Расписания и запуски заданий (Batch job schedules and runs)
  + Ввод в эксплуатацию расчетов урегулирований (Commissioning calculations settlements)
  + Быстрое и подробное назначение цен (Quick and detailed quote)
  + Иллюстрация преимущества (Benefit illustration)
  + Валидация суммарной выгоды (Benefit summary validation)

Образцы тестов для страховой заявки:

* Проверка правил претензий (Validate claims rule)
* Убедитесь, что претензия может возникнуть на максимальный и минимальный платеж (Ensure that claim can occur to the maximum and minimum payment)
* Убедитесь, что данные передаются точно во все подсистемы, включая учетные записи и отчетность (Verify data is transferred accurately to all sub-systems including accounts and reporting)
* Убедитесь, что претензии могут быть обработаны по всем каналам, например, через Интернет, мобильный телефон, звонки и т. Д (Check that the claims can be processed via all channels example web, mobile, calls, etc)
* Тест на 100% покрытие и точность в расчетах, определяющих ставки премии ( Test for 100% coverage and accuracy in calculations determining premium rates)
* Убедитесь, что формула для расчета дивидендов и выплаченных значений дает правильное значение (Make sure formula for calculating dividend and paid up values gives correct value)
* Убедитесь, что значения выдачи рассчитываются в соответствии с требованиями политики (Verify surrender values are calculated as per the policy requirement)
* Проверьте фидуциарные детали и требования бухгалтерского учета (Verify fiduciary details and bookkeeping requirements)
* Тестирование сложных сценариев для отклонения политики провала и восстановления (Test complex scenarios for policy lapse and revivals)
* Испытайте различные условия для стоимости без конфискации (Test various conditions for non-forfeiture value)
* Тестовые сценарии для прекращения действия политики (Test scenarios for policy termination)
* Убедитесь, что учетная запись главной книги ведет себя так же, как и для сверки с дополнительной книгой (Verify general ledger account behave same as to reconcile with subsidiary ledger)
* Тестовый расчет чистого обязательства для оценки (Test calculation of net liability for valuation)
* Условия тестирования для длительного страхования (Test conditions for extended term insurance)
* Проверка политики для варианта без конфискации (Verify policy for a non-forfeiture option)
* Проверьте, что другой страховой продукт ведет себя как ожидалось (Check different insurance product term behaves as expecte)
* Проверьте сумму выплаты согласно плану продукта (Verify premium value as per product plan)
* Тестирование автоматической системы обмена сообщениями для информирования клиентов о новых продуктах (Test automatic messaging system to inform customer about new products)
* Проверяйте все данные, введенные пользователями, по мере их прохождения через рабочий процесс, чтобы инициировать предупреждения, соответствие, уведомления и другие события рабочего процесса (Validate all the data entered by users as it progresses through the workflow to trigger warnings, compliance, notification and other workflow events)
* Убедитесь, что шаблон страхового документа поддерживает такой формат документа, как MS-Word (Verify insurance document template supports the document format like MS-Word)
* Тестовая система для автоматического выставления счета и отправки его клиенту по электронной почте (Test system for generating invoice automatically and send it to customer through e-mail)

**Тестирование в сфере телекоммуникаций (Telecom)**

После перехода сектора телекоммуникаций на цифровые и компьютерные сети, телекоммуникационная отрасль повсеместно использует ПО. Сектор телекоммуникаций зависит от различных видов компонентов ПО, чтобы доставить множество услуг, таких как маршрутизация и коммутация, VoIP и широкополосный доступ, и т. д. Таким образом, тестирование ПО Telecom является неизбежным.

Для предоставления телекоммуникационных услуг требуется наличие IVR, колл-центров, выставление счетов и т. д. и системы, которые включают в себя маршрутизаторы, коммутаторы, сотовые вышки и т. д.

Примеры тест-кейсов:

* Биллинговая Система:
  + Телефонный номер клиента зарегистрирован на данного оператора связи
  + Продолжает ли номер работать
  + Введенный номер является валидным, а это 10-значный номер
  + Отображение неоплаченных счетов
  + Проверьте, что все предыдущие аккаунты номера стерты
  + Убедитесь, что система зафиксировала количество звонков точно
  + Проверьте, что тариф, выбранный клиентом, отображается в биллинговой системе
  + Общая суммы расходов является точной
* Тестирование Приложения
  + Протоколы, подача сигнала, полевые испытания для IoT
  + Функциональное тестирование для базового применения мобильных телефонов как звонок, SMS, перевод/удержание и т. д.
  + Тестирование различных приложений, таких как финансы, спорт и на основе определения местоположения, и т. д. ОСС-БСС тестирования
* ОСС-БСС тестирования (OSS-BSS testing)
  + Выставление счетов клиентам, партнерам, правопорядка и борьбы с мошенничеством, обеспечения доходов
  + Сетевое управление, посредничество, обеспечение, и т. д.
  + ЦОВ, CRM и ERP-систем, хранилищ данных и т. д.
* Тестирование соответствия
  + Совместимость электрических интерфейсов
  + Соответствие протокола
  + Соответствие транспортных слоев
* Тестирование IVR
  + Интерактивные тестовые сценарии
  + Распознавание голоса
  + Голосовое меню и ветвление
  + Ввод тонового сигнала DTMF

**Тестирование протокола: L2 и L3 OSI**

Когда компьютеры общаются друг с другом, существует общий набор правил и условий, которым должен следовать каждый компьютер. Другими словами, протоколы определяют, как данные передаются между вычислительными устройствами и по сетям.

PROTOCOL testing проверяет протоколы связи в областях коммутации, беспроводной связи, VoIP, маршрутизации и т. д. Цель состоит в том, чтобы проверить структуру пакетов, которые отправляются по сети, с помощью инструментов тестирования протоколов.

Протоколы делятся на две категории: Routed и routing. Routed могут использоваться для отправки пользовательских данных из одной сети в другую. Он переносит пользовательский трафик, такой как электронная почта, веб-трафик, передача файлов и т. д. Routed являются IP, IPX и AppleTalk.

Routing это сетевые протоколы, которые определяют маршруты для маршрутизаторов. Они используется только между маршрутизаторами. Например, RIP, IGRP, EIGRP и т. д.

Модель OSI имеет в общей сложности 7 уровней сетевого взаимодействия, в которых уровень 2 и уровень 3 очень важны.

* Уровень 2: это уровень канала передачи данных. Mac-адрес, Ethernet, Token Ring и Frame Relay являются примерами канального уровня.
* Уровень 3: это сетевой уровень, который определяет наилучший доступный путь в сети для связи. IP-адрес является примером layer3.

Для тестирования протокола вам понадобится анализатор протокола и симулятор.

Анализатор протокола обеспечивает правильное декодирование наряду с анализом вызовов и сеансов. В то время как симулятор имитирует различные сущности сетевого элемента.

Обычно тестирование протокола выполняется DUT (тестируемым устройством) для других устройств, таких как коммутаторы и маршрутизаторы, и для настройки протокола в нем. После этого проверяется структура пакетов, отправленных устройствами. Он проверяет масштабируемость, производительность, алгоритм протокола и т. д. устройства с помощью таких инструментов, как lxNetworks, Scapy и Wireshark.

Тестирование протокола включает тестирование функциональности, производительности, стека протоколов, функциональной совместимости и т. д. Во время тестирования протокола, в основном, выполняется три проверки:

* Корректность: получаем ли мы пакет X как ожидали
* Задержка: сколько времени занимает доставка пакета
* Пропускная способность: сколько пакетов мы можем отправить в секунду

Тестирование протокола может быть разделено на две категории. Стресс и тесты надежности и функциональные тесты. Стресс-тесты и тесты надежности охватывают нагрузочное тестирование, стресс-тестирование, тестирование производительности и т. д. В то время как функциональное тестирование включает в себя негативное тестирование, тестирование на соответствие, тестирование на совместимость и т. д.

Тестирование соответствия: протоколы, реализованные в продуктах, тестируются на соответствие, например, IEEE, RFC и т. д. Тестирование совместимости: проверяется совместимость для разных поставщиков. Это тестирование проводится после тестирования соответствия на соответствующей платформе. Проверка функциональности сети: функциональность сетевых продуктов проверена на функциональность со ссылкой на проектную документацию. Например, функциями могут быть защита портов на коммутаторе, ACL на маршрутизаторе и т. д.

Вот примеры Test case для роутеров:

* One VLAN on One Switch: Создайте две разные VLAN. Проверьте видимость между хостами в разных VLAN
* Three Symmetric VLANs on One switch: Создайте три разных VLAN. Проверьте видимость между хостами
* Spanning Tree: Root Path Cost Variation: Проверьте, как изменяется стоимость маршрута корневого пути после изменения топологии
* Spanning Tree: Port Blocking: Проверьте, как протокол связующего дерева предотвращает образование циклов в сети, блокируя избыточные каналы, также при наличии VLAN
* Spanning Tree: Port Blocking: Покажите, что каждый MSTI может иметь разные корневые мосты
* Visibility between different STP Regions: с теми же VLAN проверить видимость между различными регионами STP
* Telephone switch Performance: Создайте 1000 телефонных звонков и проверьте, нормально ли работает телефонный коммутатор или его производительность снижается
* Negative test for device: Введите неверный ключ и проверьте пользователя на аутентификацию.
* Line speed: Проверьте устройство, работающее на скорости 10 Гбит / с, используя всю доступную пропускную способность для обработки входящего трафика.
* Protocol conversation rate: Отслеживайте диалог TCP между двумя устройствами и убедитесь, что каждое устройство работает правильно
* Response time for session initiation: Измерьте время отклика устройства на запрос приглашения для инициации сеанса

**Тестирование интернета вещей (IoT — Internet of Things)**

Интернет вещей — это сеть, состоящая из устройств в транспортных средствах, зданиях или любых других подключенных электронных устройств. Эта взаимосвязь облегчает сбор и обмен данными. 4 общих компонента системы IoT:

* Sensor
* Application
* Network
* Backend (Data Center)

IOT — это соединение идентифицируемых встроенных устройств с существующей интернет-инфраструктурой. Проще говоря, мы можем сказать, что IOT - это эра «умных», связанных продуктов, которые обмениваются данными и передают большой объем данных и загружают их в облако.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| IOT elements Testing Types | Sensor | Application | Network | Backend (Data Center) |
| Functional testing | True | True | False | False |
| Usability testing | True | True | False | False |
| Security testing | True | True | True | True |
| Performance testing | False | True | True | True |
| Compatibility testing | True | True | False | False |
| Services testing | False | True | True | True |
| Operational testing | True | True | False | False |

Категории тестов с примерами Test Conditions:

* Проверка компонентов (Components Validation):
  + Аппаратное обеспечение устройства (Device Hardware)
  + Встроенное программное обеспечение (Embedded Software)
  + Облачная инфраструктура (Cloud infrastructure)
  + Подключение к сети (Network Connectivity)
  + Стороннее программное обеспечение (Third-party software)
  + Тестирование датчиков (Sensor testing)
  + Тестирование команд (Command testing)
  + Тестирование формата данных (Data format testing)
  + Испытание на прочность (Robustness testing)
  + Тестирование безопасности (Safety testing)
* Проверка функций (Function Validation):
  + Базовое тестирование устройства (Basic device testing)
  + Тестирование между устройствами IOT (Testing between IOT devices)
  + Обработка ошибок (Error Handling)
  + Правильность расчета (Valid Calculation)
* Проверка соответствия (Conditioning Validation):
  + Ручная (Manual Conditioning)
  + Автоматическая (Automated Conditioning)
  + Профили (Conditioning profiles)
* Проверка производительности (Performance Validation):
  + Частота передачи данных (Data transmit Frequency)
  + Обработка многократнах запросов (Multiple request handing)
  + Синхронизация (Synchronization)
  + Тестирование прерываний (Interrupt testing)
  + Производительность устройства (Device performance)
  + Проверка согласованности (Consistency validation)
* Безопасность и проверка данных (Security and Data Validation):
  + Проверка пакетов данных (Validate data packets)
  + Проверка на потерю или повреждение пакетов (Verify data loses or corrupt packets)
  + Шифрование / дешифрование данных (Data encryption/decryption)
  + Значения данных (Data values)
  + Роли и ответственность пользователей и их модель использования (Users Roles and Responsibility & its Usage Pattern)
* Проверка шлюза:
  + Тестирование облачного интерфейса (Cloud interface testing)
  + Тестирование протокола от устройства к облаку (Device to cloud protocol testing)
  + Тестирование задержек (Latency testing)
* Проверка аналитики (Analytics Validation):
  + Проверка аналитики данных датчика (Sensor data analytics checking)
  + Операционная аналитика системы IOT (IOT system operational analytics)
  + Аналитика системного фильтра (System filter analytics)
  + Проверка правил (Rules verification)
* Проверка связи (Communication Validation):
  + Совместимость (Interoperability)
  + M2M или от устройства к устройству (M2M or Device to Device)
  + Тестирование трансляции (Broadcast testing)
  + Тестирование прерываний (Interrupt testing)
  + Протокол (Protocol)

**Что такое облачное тестирование? (Cloud testing)**

CLOUD testing — это тип тестирования программного обеспечения, который проверяет услуги облачных вычислений. Облачные вычисления - это интернет-платформа, предоставляющая различные компьютерные сервисы, такие как оборудование, программное обеспечение и другие компьютерные сервисы, удаленно. Существует три модели облачных вычислений:

* SaaS- Software as a service
* PaaS- Platform as a service
* IaaS- Infrastructure as a service

Все облачное тестирование разделено на четыре основные категории:

* Тестирование всего облака (Testing of the whole cloud). Облако рассматривается как единое целое и на основе его возможностей проводится тестирование. SaaS и облачные вендоры, а также конечные пользователи заинтересованы в проведении такого типа тестирования.
* Тестирование в пределах облака (Testing within a cloud). Проверяя каждую из его внутренних функций, проводится тестирование. Только поставщики облачных услуг могут выполнять этот тип тестирования.
* Тестирование через облако (Testing across cloud). Тестирование проводится в облачных, частных, публичных и гибридных облаках различных типов.
* SaaS-тестирование в облаке (SaaS testing in cloud): функциональное и нефункциональное тестирование проводится на основе требований приложений.

Облачное тестирование фокусируется на основных компонентах, таких как:

* Приложение (Application): охватывает тестирование функций, сквозные бизнес-процессы (end-to-end business workflows), безопасность данных, совместимость с браузерами и т. д.
* Сеть (Network): включает в себя тестирование различной пропускной способности сети, протоколов и успешную передачу данных через сети.
* Инфраструктура (Infrastructure): включает в себя тестирование аварийного восстановления, резервное копирование, безопасное соединение и политики хранения. Инфраструктура должна быть проверена на соответствие нормативным требованиям.

Другие типы тестирования в облаке включают:

* Performance
* Availability
* Compliance
* Security
* Scalability
* Multi-tenancy
* Live upgrade testing

Как выполнять облачное тестирование:

* SaaS или облачное тестирование: Этот тип тестирования обычно выполняется поставщиками облачных или SaaS-приложений. Основной задачей является обеспечение качества предоставляемых сервисных функций, предлагаемых в облачной или SaaS-программе. Тестирование, выполняемое в этой среде, — это проверка интеграции, функциональности, безопасности, функциональности модулей, системных функций и регрессионного тестирования, а также оценка производительности и масштабируемости.
* Онлайн тестирование приложений в облаке: Производители онлайн-приложений проводят это тестирование, которое проверяет производительность и функциональное тестирование облачных сервисов. Когда приложения связаны с legacy системами, проверяется качество связи между legacy системой и тестируемым приложением в облаке.
* Тестирование облачных приложений над облаками: Для проверки качества облачного приложения в разных облаках выполняется этот тип тестирования.

Примеры Test Scenario и несколько Test case для каждого из них:

* Тестирование производительности ([Performance testing](https://www.guru99.com/performance-testing.html)):
  + Сбой из-за одного действия пользователя в облаке не должен влиять на других пользователей
  + Ручное или автоматическое масштабирование не должно вызывать сбоев
  + На всех типах устройств производительность приложения должна оставаться неизменной
  + Повторное бронирование на стороне поставщика не должно снижать производительность приложения
* Тестирование безопасности (Security testing):
  + Только авторизованный клиент должен получать доступ к данным
  + Данные должны быть хорошо зашифрованы
  + Данные должны быть полностью удалены, если они не используются клиентом
  + Администрация поставщиков не должна получать доступ к данным клиентов.
  + Проверьте наличие различных настроек безопасности, таких как брандмауэр, VPN, антивирус и т. д.
* Функциональное тестирование (Functional testing):
  + Валидный ввод должен давать ожидаемые результаты
  + Сервис должен должным образом интегрироваться с другими приложениями
  + Система должна отображать тип учетной записи клиента при успешном входе в облако
  + Когда клиент решил переключиться на другие службы, работающая служба должна автоматически закрыться
* Тестирование совместимости (Interoperability & Compatibility testing):
  + Проверка требований совместимости тестируемой системы и приложения
  + Проверьте совместимость браузера в облачной среде
  + Определите дефект, который может возникнуть при подключении к облаку
  + Любые неполные данные в облаке не должны быть переданы
  + Убедитесь, что приложение работает на другой платформе облака
  + Протестируйте приложение в собственной среде, а затем разверните его в облачной среде.
* Тестирование сети (Network testing):
  + Тестовый протокол, отвечающий за подключение к облаку
  + Проверка целостности данных при передаче данных
  + Проверьте правильность подключения к сети
  + Проверьте, отбрасываются ли пакеты брандмауэром с обеих сторон
* Нагрузка и стресс-тестирование (Load and Stress testing):
  + Проверьте сервисы, когда несколько пользователей получают к ним доступ
  + Определите дефект, ответственный за сбой оборудования или среды
  + Проверьте, отказывает ли система при увеличении удельной нагрузки
  + Проверьте, как система изменяется со временем при определенной нагрузке

**Что такое тестирование сервис-ориентированной архитектуры? (SOA — Service Oriented Architecture)**

Это тестирование архитектурного стиля SOA, в котором компоненты приложения предназначены для сообщения по протоколам связи, обычно через сеть. SOA — это метод интеграции бизнес-приложений и процессов для удовлетворения потребностей бизнеса. В разработке программного обеспечения SOA обеспечивает гибкость бизнес-процессов. Изменения в процессе или приложении могут быть направлены на конкретный компонент, не затрагивая всю систему. Разработчики программного обеспечения в SOA либо разрабатывают, либо покупают куски программ под названием SERVICES. Что такое Service?

* Service могут быть функциональной единицей приложения или бизнес-процесса, которая может быть повторно использована или повторена любым другим приложением или процессом. (Например, Платежный шлюз — это сервис, который может быть повторно использован любым сайтом электронной коммерции. Каждый раз, когда необходимо сделать платеж, сайт электронной коммерции вызывает / запрашивает сервис Платежного шлюза. После оплаты через шлюз, ответ отправляется на сайт электронной коммерции)
* Service просты в сборке и легко переконфигурируют компоненты.
* Service можно сравнить со строительными блоками. Они могут построить любое необходимое приложение. Добавить и удалить их из приложения или бизнес-процесса очень просто.
* Service больше определяются бизнес-функциями, которые они выполняют, а не кусками кода.

Пример: на домашней странице веб-сайта и в поисковой системе отображается ежедневный прогноз погоды. Вместо того, чтобы писать код для виджета прогноза погоды, у продавца можно купить Службу прогноза погоды и встроить ее в страницу.

Тестирование SOA должно быть сосредоточено на 3 уровнях:

* Уровень сервисов (Services Layer): Этот уровень состоит из сервисов, полученных из бизнес-функций. Например — Рассмотрим оздоровительный сайт, который состоит из: Трекер веса, Отслеживание уровня сахара в крови и трекера артериального давления. Трекеры отображают соответствующие данные и дату их ввода. Уровень сервисов состоит из сервисов, которые получают соответствующие данные из базы данных – Сервис трекера веса, сервис отслеживания уровня сахара в крови, сервис отслеживания артериального давления и Сервис логина.
* Уровень процесса (Process Layer): Уровень процесса состоит из процессов, набора сервисов, которые являются частью единой функциональности. Процессы могут быть частью пользовательского интерфейса (например, поисковая система), частью инструмента ETL (для получения данных из базы данных). Основное внимание на этом уровне будет уделяться пользовательским интерфейсам и процессам. Пользовательский интерфейс весового трекера и его интеграция с базой данных является основным направлением.
* Потребительский уровень (Consumer Layer): Этот уровень в основном состоит из пользовательских интерфейсов. Тестирование приложения SOA распределяется на три уровня: Уровень обслуживания, Уровень интерфейса, Уровень end-to-end. Подход сверху вниз используется для проектирования тестов. Подход снизу-вверх используется для выполнения теста.

Методы тестирования SOA:

* Data based testing на основе бизнес-сценариев:
  + Различные аспекты бизнеса, связанные с системой, должны быть проанализированы.
  + Сценарии должны быть разработаны на основе интеграции
    - Различные веб-сервисы приложения
    - Веб-сервисы и приложения
  + Настройка данных должна быть выполнена на основе вышеуказанных сценариев.
  + Настройка данных должна быть выполнена так, чтобы охватить также сквозные сценарии
* Заглушки (Stubs):
  + Будут созданы фиктивные интерфейсы для тестирования сервисов.
  + Через эти интерфейсы могут быть предоставлены различные входные данные, а выходные данные могут быть проверены.
  + Когда приложение использует интерфейс с внешней службой, которая не тестируется (сторонняя служба), во время тестирования интеграции можно создать заглушку.
* Regression testing:
  + Регрессионное тестирование приложения должно проводиться при наличии нескольких релизов, чтобы обеспечить стабильность и доступность систем.
  + Будет создан комплексный набор регрессионных тестов, охватывающий сервисы, которые составляют важную часть приложения.
  + Этот набор тестов может быть повторно использован в нескольких релизах проекта.
* Тестирование уровня сервиса (Service Level testing):
  + Тестирование уровня сервиса включает тестирование компонента на функциональность, безопасность, производительность и функциональную совместимость. Каждую услугу необходимо сначала протестировать независимо.
* Functional testing:
  + Убедитесь, что служба отправляет правильный ответ на каждый запрос.
  + Правильные ошибки получены для запросов с неверными данными.
  + Проверьте каждый запрос и ответ для каждой операции, которую служба должна выполнять во время выполнения.
  + Проверяйте сообщения об ошибках при возникновении ошибки на уровне сервера, клиента или сети.
  + Убедитесь, что полученные ответы имеют правильный формат.
  + Подтвердите, что данные, полученные в ответе, соответствуют запрашиваемым данным.
* Security testing:
  + Отраслевой стандарт, определенный тестированием WS-Security, должен соблюдаться веб-службой.
  + Меры безопасности должны работать без нареканий.
  + Шифрование данных и Цифровые подписи на документах
  + Аутентификация и авторизация
  + SQL-инъекции, вредоносные программы, XSS, CSRF и другие уязвимости должны быть проверены на XML. Атаки отказа в обслуживании
* Performance testing:
  + Производительность и функциональность сервиса необходимо тестировать при большой нагрузке.
  + Производительность службы необходимо сравнивать при работе индивидуально и в приложении, с которым она связана.
  + Нагрузочное тестирование сервиса: проверить время отклика, проверить наличие узких мест, проверить использование процессора и памяти, прогнозировать масштабируемость
* Тестирование уровня интеграции (Integration level testing):
  + Интеграционное тестирование проводится с упором в основном на интерфейсы.
  + Этот этап охватывает все возможные бизнес-сценарии.
  + Нефункциональное тестирование приложения должно быть сделано еще раз на этом этапе. Security, compliance, and Performance testing обеспечивают доступность и стабильность системы во всех аспектах.
  + Коммуникационные и сетевые протоколы должны быть протестированы для проверки согласованности обмена данными между сервисами.
* End to End testing:
  + Все сервисы работают должным образом после интеграции
  + Обработка исключений
  + Пользовательский интерфейс приложения
  + Правильный data flow через все компоненты
  + Бизнес-процесс

**Что такое тестирование планирования ресурсов предприятия? (ERP — Enterprise Resource Planning)**

Планирование ресурсов предприятия, также известное как ERP, представляет собой комплексное программное обеспечение, которое объединяет различные функции организации в единую систему. Программное обеспечение имеет общую базу данных, содержащую всю информацию, относящуюся к различным функциям или подразделениям организации. Система ERP помогает оптимизировать процессы и доступ к информации по всей организации 24 × 7.

Приложения ERP стали критически важными для бесперебойной работы предприятий. Поскольку они включают в себя множество модулей, функций и процессов, необходимость их проверки становится критической. Предприятия осознают необходимость использования модели SMAC (Social, Mobile, Analytics и Cloud) для ускорения роста. Однако капитальный ремонт основных процессов, администрируемых устаревшими приложениями ERP, также важен. Приложения ERP помогают предприятиям управлять различными функциями, отделами и процессами, включая генерируемые в них данные. Эти приложения помогают предприятиям работать как единое целое и в процессе генерировать такие результаты, как повышение производительности, повышение эффективности, сокращение отходов, повышение качества обслуживания клиентов и повышение рентабельности инвестиций. Ввиду важности приложений ERP для организаций, они должны быть протестированы и утверждены. Тестирование приложений ERP может обеспечить бесперебойную работу множества задач в организациях. Они могут включать в себя отслеживание инвентаризации и операций с клиентами, управление финансами и человеческими ресурсами, среди многих других.

Каждое программное обеспечение ERP поставляется с несколькими версиями и требует настройки в соответствии с конкретными бизнес-требованиями. Более того, поскольку каждый элемент приложения связан с каким-либо другим модулем, их обновление может быть сложной задачей. Например, для создания заказа на продажу потребуется доступ к модулю управления запасами. Если какой-либо из модулей не функционирует оптимально, это может повлиять на все приложение ERP. Это может оказать каскадное влияние на производительность компании, а также создать плохой опыт клиентов. Следовательно, тестирование приложений ERP должно обеспечить правильную реализацию программного обеспечения и предотвратить сбои. Тестирование программного обеспечения ERP, помимо проверки функциональности программного обеспечения, должно обеспечивать точное формирование отчетов и форм. Выявляя и устраняя ошибки на этапе тестирования, тестировщики могут избежать столкновения с проблемами после внедрения. Более того, это может привести к скорейшему внедрению программного обеспечения и обеспечить его бесперебойную работу. Службы тестирования приложений ERP проверяют бизнес-процессы, функции и регулирующие их правила. Они помогают снизить операционные риски в условиях ограниченности имеющихся ресурсов и времени.

Поскольку система ERP содержит огромные объемы данных, тестирование ручных процедур может потребовать много времени и средств. Автоматизированное тестирование может помочь проверить все функции и возможности при минимальных затратах времени и средств. Кроме того, поскольку несколько бизнес-подразделений организации могут иметь разные процессы или процедуры, автоматическое тестирование может проверять точность их результатов по конкретным параметрам. Кроме того, ERP-систему необходимо периодически обновлять с появлением новых технологий, таких как Cloud, Big Data и Mobility. Такие обновления помогают организации проверять транзакции в режиме реального времени, что невозможно вручную.

Системы ERP доступны в нескольких версиях, предназначенных для нескольких доменов, подразделений и клиентов, лучшие доступные инструменты:

* Microsoft Dynamics NAV — для малых и средних предприятий
* SAP Insurance — Для страховых компаний
* Microsoft Dynamics AX — для крупных предприятий
* SAP Banking — Для банковского сектора

**Тестирование качества видеосвязи WebRTC-based сервиса видеоконференций**

WebRTC (англ. real-time communications — коммуникации в реальном времени) — проект с открытым исходным кодом, предназначенный для организации передачи потоковых данных между браузерами или другими поддерживающими его приложениями по технологии точка-точка. Стандарт поддерживается в браузерах, поэтому низкий порог вхождения – не нужно писать клиент. Помимо браузеров известны такие гиганты в сфере видеоконференций, как: Skype, Google Meets/hangouts, Discord.

В чем выражается качество видеосвязи? В подавляющем большинстве случаев речь о:

* Разрешение
* Количество кадров в секунду

Как обычно пытаются тестировать? С помощью плохой сети. Например, отойти с планшетом подальше от wi-fi точки. Вообще плохая сеть подразумевает большой пинг, низкую пропускную способность канала, потерю пакетов.

К сожалению, ручное тестирование видеосвязи (впрочем, как и аудио-) не даст достоверных и точных результатов. На следующем этапе команда приходит к идее писать автотесты (по большей части unit) и лишь некоторые доходят до написания бенчмарков. Возможно, в комментариях опытные коллеги поделятся своим опытом.

**Что такое тестирование ETL?**

ETL расшифровывается как Extract, Transform, Load. ETL — это процесс, объединяющий три этапа: извлечение, преобразование и загрузка данных из одного источника в другой. Проще говоря, операции ETL выполняются с данными, чтобы вытащить их из одной базы данных в другую. Процесс ETL часто используется в хранилищах данных.

* Первым этапом процесса ETL является извлечение данных. На этом этапе данные извлекаются из исходной базы данных; может быть более одного источника данных.
* На втором этапе, Преобразование данных, извлеченные данные преобразуются путем применения различных правил и функций, которые должны храниться в целевой базе данных в надлежащем формате. Данные извлекаются из разных источников, и вполне вероятно, что у них будет много проблем, например, одному и тому же объекту присваиваются разные имена или одно и то же имя присваивается разным объектам.
* На последнем этапе загрузка данных, преобразованные и однородно отформатированные данные загружаются в базу данных назначения.

ETL-тестирование — это тип тестирования, выполняемый для гарантии того, что данные, перенесенные из исходной в целевую базу данных, являются точными и соответствуют действующим правилам преобразования.

Пример:

Давайте рассмотрим пример слияния двух компаний — компании A и компании B. После слияния их операции будут объединены, а их клиенты, сотрудники и другие данные будут храниться в единой централизованной базе данных. Предположим, что компания A использует базу данных Oracle для хранения всей информации, а компания B использует MySQL. Теперь для объединения своей информации обе компании могут использовать процесс ETL для переноса данных из своих отдельных баз данных в одну согласованную базу данных. В процессе ETL, поскольку две базы данных различны, данные обеих компаний будут в другом формате, будут использоваться разные соглашения об именах, будут использоваться разные структуры таблиц и так далее. Из-за этих различий компаниям необходимо удостовериться, что перед загрузкой данных в целевую базу данных она была должным образом очищена и может сформировать нужный формат. При тестировании ETL тестеры должны убедиться, что данные обеих баз данных были преобразованы в формат целевой базы данных; необходимые функции преобразования были выполнены; в процессе не было потеряно никаких данных, и данные являются точными.

Типы тестирования ETL:

* Новое тестирование хранилища данных (New Data Warehouse Testing) — в этом типе тестирования ETL все делается с нуля. Информация для ввода данных собирается от клиента. Исходные и целевые базы данных заново создаются и проверяются с использованием инструментов ETL.
* Миграционное тестирование (Migration Testing) — в этом типе тестирования ETL у клиента есть существующее хранилище рабочих данных; у клиента также есть существующий инструмент ETL. Процесс тестирования миграции требуется, когда данные загружаются из существующей базы данных в новую базу данных. Старая база данных называется устаревшей или исходной базой данных, а новая база данных называется целевой базой данных.
* Запрос на изменение (Change Request) — в этом процессе данные выбираются из разных источников и загружаются в существующее хранилище, при этом не используются никакие новые базы данных. Помимо загрузки новых данных, клиенту может потребоваться изменить существующее бизнес-правило или добавить новое бизнес-правило.
* Тестирование отчетов (Report Testing). После создания хранилища данных система позволяет пользователям создавать различные отчеты. Это тестирование проверяет макет, точность данных и ограничения доступа пользователей к отчетам.

**Мобильное тестирование**

**Каковы особенности в тестировании мобильных приложений?**

* Высокая фрагментация устройств (посмотреть: лучшеаналитика с ваших пользователей, но есть и источники статистики по странам, версиям ОС и вендорам)
* Размеры дисплеев, разрешение и сам по себе тач-интерфейс (ландшафтная и портретная ориентация, все элементы должны быть такого размера, чтобы можно было однозначно по ним попасть; сценарии не должны вести на пустые экраны. Всегда нужно проверять несколько нажатий на одну и ту же кнопку, мультитач (если поддерживается, если нет то аппа не должна на него реагировать), нативные жесты.
* Постоянная обратная связь с пользователем (реакция кнопок на нажатие - у каждого элемента должно быть нажатое состояние, должны быть сообщения при загрузке контента/прогресс, сообщения об успешном исполнении сценария, звук\вибрация должны быть корректно синхронизированы по событию, сообщения при ошибке доступа к сети, наличие сообщений при попытке удалить важную информацию, наличие экрана/сообщения при окончании процесса/игры (экран Game over)).
* Ограничения ресурсов (энергопотреблление, утечки памяти (особенно может проявляться на окнах, с большим количеством информации (длинные списки как пример), во время задач с длительным workflow (когда пользователь долго не выходит из приложения), при некорректно работающем кэшировании изображений), не хватает места для установки и работы, обновления, перенос на SD карту.
* Реакция на внешние прерывания (выключение или перезагрузка, входящий звонок или сообщение, обновления приложений, уведомления, разрядка, переход в энергосберегающий режим и режим ожидания, смена ориентации + в режиме ожидания, переход wi-fi -> 3G и обратно, включение и отключение функций необходимых устройству (геопозиции, блютус, режим полета), запрет на использование аппаратных ресурсов, функции с камерой, кейсы с потерей связи, зарядкой устройства, подключением отключением сд карты/симки/АКБ, подключение кабеля или гарнитуры, биометрические (напр для банковского приложения), платежи NFC или просто разные фичи, сворачивание приложения, принудительная остановка).
* Если приложение работает с какими-то форматами файлов, нужно проверять корректность работы с каждым из них
* Учитывать шторку и челку или вырез под фронталки
* Бэкап и восстановление из него
* Работа в режиме разделенного экрана
* Датчики, температура окружающей среды
* Тестирование смартфона начинается с тестирования самого устройства в заводском состоянии.

**В чем отличия тестирования мобильного приложения от десктопного?**

* В первую очередь это разные операционные системы и разная архитектура «железа», хотя сейчас прогресс нацелен на унификацию (например, такие гиганты, как Microsoft и Apple. У MS это планшеты-ноутбуки Surface на базе ARM и Windows 10, Apple в июне 2020 года заявила о переходе на ARM-архитектуру в компьютерах).
* Пока еще актуальное различие – аппаратные ресурсы. Мощность начинки и количество памяти. Хотя, опять же, в 2020 году этот пункт уже утрачивает актуальность. Мощности новых флагманских мобильных устройств сопоставимы с бюджетными ноутбуками на архитектуре x86/64, а начинка бюджетных и средних по цене мобильных аппаратов обеспечивает достаточный для большинства нужд уровень производительности.
* Самое очевидное различие в аппаратной части помимо мощности – наличие разнообразных датчиков и модулей связи в мобильном устройстве, а также нескольких камер, вибромотора, сканера отпечатков и т. д.
* Наличие датчика ориентации уже предполагает тестирование в портретной и ландшафтной ориентациях. Добавьте к этому можество разрешений дисплея, их различиные типы матриц со своими особенностями отображения и т.п.
* Помимо этого, в мобильном устройстве очень большое внимание уделяется обработке разнообразных прерываний.
* Основная функция мобильных устройств – по-прежнему связь. Голосовая, но также и через мобильный интернет, что усложняет тестирование по сравнению с десктопами.
* Связь также в контектсте взаимодействия с носимой электроникой (беспроводные наушники, фитнес-браслеты, смарт-часы, очки и т.п.), бесконтактной оплатой и т.п.
* Прогресс в обновлениях ОС. В мобильных устройствах это происходит гораздо чаще и имеет большее значение в связи с большой конкуренцией.
* Различия в гайдлайнах для ОС.
* Десктопные приложения чаще всего загружаются с официального веб-сайта производителя. Мобильное приложения почти всегда загружается из соответствующего ОС магаза приложений.
* Ожидания. Т.к. приложения десктопных устройств созданы в основном для осуществления некой функциональной деятельности и являются рабочими инструментами, им может быть иногда простительно то, что в мобильном приложении приведет к немедленному удалению его пользователем и негативным отзывам. Любая мелкая проблема с интуитивностью, интерфейсом, локализацией, функциональностью, производительностью, расходом батери может моментально отпугнуть пользователя и тот отдаст предпочтение конкурентам.

**В чем отличия тестирования мобильного приложения от web?**

* Большее количество вариантов и комбинаций ОС/железа и т.п. в мобильных устройствах (сюда же вытекающее следствие необходимости использования эмуляторов)
* Браузеры «стационарны», в то время как при тестировании мобильных приложений нужно учитывать ориентацию, прерывания, связь, наличие дополнительных модулей.
* С т.з. связи веб приложение фактически становится бесполезным при потере интернет-соединения (хотя в последнее время это иногда не совсем так), для нативного мобильного приложения ничего не изменится\*.
* Аппаратные ограничения. Веб приложению обычно доступно куда больше ресурсов.
* Публикация и распространение. Для того, чтобы люди начали пользоваться мобильным приложением, необходим аккаунт разработчика и пройденная модерация в магазине приложений.

Из этих основных различий следуют и различия в тестировании – уровни, виды, типы и т. д.

**Почему так много внимания уделяется прерываниям? Что такое Activity Lifecycle?**

У приложения есть жизненный цикл со строго определенными в системе активностями. Данные активити должны корректно обрабатываться.

На просторах много материала по данной теме, возможно позже оформлю здесь подробнее.

**Что вы знаете о симуляторах и эмуляторах?**

Всегда идет спор о том, что тестирование мобильных приложений более экономично и быстрее на реальных устройствах. Но прежде чем разрушить этот миф, давайте взглянем на базовое определение эмуляторов, симуляторов и реальных устройств:

* Тестирование на реальном устройстве позволяет запускать мобильные приложения и проверять его функциональность. Тестирование реального устройства гарантирует, что ваше приложение будет работать без проблем на клиентских телефонах.
* Эмулятор — это программа, которая позволяет вашему мобильному устройству имитировать функции другого компьютера или мобильного программного обеспечения, которое вы хотите имитировать, устанавливая их на свой компьютер или мобильный телефон. Эмулятор нацелен на эмуляцию или имитацию, насколько это возможно, внешнего поведения объекта. Эмуляторы предпочтительны, когда команде тестирования необходимо проверить только внешнее поведение мобильного телефона.
* Задача симулятора состоит в том, чтобы моделировать внутреннее устройство объекта как можно ближе.

Тестирование на симуляторах или эмуляторах имеет много преимуществ, так как они бесплатны, имеют высокую скорость выполнения, не имеют проблем при автоматическом тестировании, просты в отладке и просты в настройке. Но эмулятор / симулятор не всегда является лучшим решением для таких сценариев, как те, в которых команде тестирования необходимо проверять производительность приложения в течение более длительного периода времени. Реальные устройства стоят дороже по сравнению с эмулятором / симуляторами. Но, как следует из названия, результаты реальных устройств всегда более точны по сравнению с эмуляторами или симуляторами. Для тестирования мобильных приложений всегда требуется большое количество мобильных устройств, поскольку тестирование мобильных приложений — это все о комбинациях ПО и железа. Следовательно, в таких случаях эффективность может быть достигнута только с помощью реальных устройств.

**Типы мобильных приложений?**

* Нативные приложения: Эти приложения называют нативными оттого, что они написаны на родном (с англ. native – родной) для определённой платформы языке программирования. Для Android этим языком является Kotlin/Java, тогда как для iOS – objective-С или Swift. Нативные приложения находятся на самом устройстве, доступ к ним можно получить, нажав на иконку. Они устанавливаются через магазин приложений (Play Market на Android, App Store на iOS и др.). Они разработаны специально для конкретной платформы и могут использовать все возможности устройства – камеру, GPS-датчик, акселерометр, компас, список контактов и всё остальное. Также они могут распознавать стандартные жесты, предустановленные операционной системой или совершенно новые жесты, которые используются в конкретном приложении. В силу того, что нативные приложения оптимизированы под конкретную ОС, они органично вписываются в любой смартфон, отличаясь высокой скоростью работы и производительностью. Нативные приложения могут получить доступ к системе оповещений устройства, а также, в зависимости от предназначения нативного приложения, оно может всецело или частично обходиться без наличия интернет-соединения.
* Мобильные веб-приложения: На самом деле мобильные веб-приложения не являются приложениями как таковыми. Ведь дело в том, что веб-приложение, в сущности, представляет собой сайт, который адаптирован и оптимизирован под любой смартфон. И для того, чтобы воспользоваться им, достаточно иметь на устройстве браузер, знать его адрес и располагать интернет-соединением (благодаря ему происходит обновление информации в данном виде приложений). Запуская мобильные веб-приложения, пользователь выполняет все те действия, которые он выполняет при переходе на любой веб-сайт, а также получает возможность «установить» их на свой рабочий стол, создав закладку страницы веб-сайта. Веб-приложения отличаются кроссплатформенностью, то есть способны функционировать, независимо от платформы девайса. Козырем в их рукаве выступает и то, что они не используют его программное обеспечение. А по причине того, что являются мобильной версией сайта с расширенным интерактивом, веб-приложения не отбирают драгоценное место в памяти смартфона. Веб-приложения стали широко популярны в то время, когда начал развиваться HTML5 и люди осознали, что могут получить доступ к множеству функций нативных приложений, просто зайдя на веб-сайт через обычный браузер. На сегодняшний день сложно сказать, где именно располагается чёткая граница между веб-приложениями и обычными веб-страницами, поскольку функционал HTML5 растёт с каждым днём и всё больше и больше сайтов его используют. В то же время камень в огород веб-приложений следует бросить за неспособность работать с ними без Интернета. Причём из этого выплывает и другой минус – их производительность, которая находится на среднем уровне, в сравнении с другими видами приложений. Более того, она зависит от возможностей интернет-соединения провайдера услуг. Также очевидно, что веб-приложения не могут получить доступ к функциям системы и самого устройства.
* Гибридные приложения представляют собой сочетание веб и нативных приложений: доступ к функционалу смартфона (API системы, пуш и т.п.), размещение в маркетах, простота обновления контента, кроссплатформенность веб-части. Фактически это можно объяснить как нативное приложение, в обертке которого загружается веб-приложение. Это может быть основным контентом или лишь одной из функций. Очевидно, без интернет-соединения соответствующая часть приложения потеряет работоспособность.

**Что основное проверить при тестировании мобильного приложения?**

Проверяем в начале тестирования:

* достаточно ли устройств для тестирования (по целевым рынкам) и готово ли покрытие
* согласованы все процедуры на проекте (понять что всё готово, выстроили регламент цикл разработки, договорились кто когда вступает в работу, набрана команда)
* аппа соответствует гайдлайнам
* аппа работает в различных окружениях
* аппа должна корректно работать с внешними и внутренними запросами
* Проверка энергопотребления (как типичный юзер)
* Выбор тестов для автоматизации (когда уже написано достаточно ручных)

Список кейсов под iPhone из утерянного источника, довольно неплохой:

* Перепроверка функциональности, где ранее были обнаружены наиболее критические дефекты (регрессионное тестирование)
* Проверка функциональности на корректных данных (текущая дата, короткие имена и т.д.)
* Проверка на некорректных значениях (например: пустые поля, длинные имена, установка на телефоне даты в прошлом и т.д.)
* Проверка интерфейса приложения на соответствие требованиям Apple (Human interface guidelines for iPhone\iPad)
* Производительность приложения и скорость ответа интерфейса (используется iPhone 2g)
* Тесты на удобство пользования приложением (Usability tests)
* Тест на совместимость с другими приложениями\функциональностью iPhone (будильник\таймер\напоминания\входящий звонок\смс)
* Проверка настроек приложения и корректность их применения
* Поиск возможных мест «падения» приложения (crash) и причин их возникновения
* Корректность работы приложения при использовании wi-fi\gprs (включая обрывы связи\ее отсутствие)
* Проверка на корректность работы приложения с памятью iPhone (memory leaks)
* проверка того, что звук не пропадает при подключении наушников
* Поведение приложения при переходе iPhone в спящий режим
* Работа приложения с акселерометром (поворот экрана в соответствии с положением iPhone, использование функции акселерометра для получения данных приложением (шагомер))
* Тестирование локализации (при поддержке приложением)
* Проверка корректности работы приложения с камерой iPhone (если такая функциональность поддерживается), а также корректность работы приложения с iPod.
* быстрые «клики» по элементам интерфейса (переход по категориям, переход по записям внутри категории)
* если есть длительный workflow – проводить его весь (вроде длинных программ в Yoga) в реальном времени
* если есть готовый список и поле для вбивания параметров, то проверить поведение, когда в поле появляется подсказка из словаря и одновременно кликаешь по записи в списке <> подсказке. возможны конфликты между подсказкой айфона и реальным выбором.
* проверка контента: адекватный размер изображений (до 1МБ) и достаточное качество. Дополнительно смотреть на iPhone4 (большее разрешение) + см. MobileHIG.pdf chapter 11 для требований к разрешению изображений.
* GUI: иконки соответствуют тому, к чему относятся (хелп – знак вопроса, настройки – шестеренка и т.д.), новые окна плавно открываются справа, присутсвует значок загрузки если происходит длительный процесс)
* Наличие экрана Game Over и корректные ссылки на нем – для игровых проектов (+ корректная отработка попадания на этот экран)

Мультиплеерные игры.

* корректность подключения игроков (напр., списывание баланса только после подключения)
* временные лаги
* подключение через различные сети
* корректное поведение при отключении игроков
* подключение ботов (если используются)

Conformance testing:

* Protocol testing
* Safety/[Security testing](https://www.guru99.com/what-is-security-testing.html)
* SIM card testing
* Radio Frequency(RF) testing
* Audio Tests
* Specific Absorption Tests

И еще: Физическая\виртуальная клавиатура, проверка обновления и чистой установки. Платный контент: соответствие цен и содержимого, покупки 2 типов (восстановливаемые и невосстанавливаемые (кредиты)) - проверка восстановления покупок привязанных к учетке (переустановка\обновление\другое устройство)+должен быть выбор из текущего прогресса и сохраненного в учетке. Реклама: не должна перекрывать элементы, должна иметь доступную кнопку закрытия (А если кнопка еще не появилась, то каунтдаун до этого). Глобализация - по геолокации или вручную меняется всё что нужно и это происходит корректно. Защита от получения преимуществ при манипуляциях с датой и временем. Копирование и вставка из\в приложение.

**Как быть с покрытием девайсов?**

Большинство багов обнаруживается на покрытии около 30% девайсов - разрешение, мощность, версия ос+нижняя граница для данной аппы. Варианты: физическая ферма устройств, эмулятор и симулятор (BrowserStack (облачный), родной в Android Studio, BlueStack, Genymotion и т.п.). Вообще физический устройств желательно иметь хотябы штук 6 - два отличающихся айфона, по планшету на iOS и Android, 2 разных андроида. Хуавей сейчас тестится отдельно из-за гугл сервисов.  
Есть хорошая статья по теме: <https://habr.com/ru/post/464433/>

**Последнее обновление Android/iOS, что нового?**

Актуализируется перед собеседованием. На данный момент в iOS 14 основное это:

* виджеты (с возможностью прокрутки нескольких на одном месте) х4\х8\х16 клеток сетки
* функция App library сортирует приложения по своим алгоритмам. Самые используемые выделяются большими иконками, остальные размещаются по папкам
* новый интерфейс входящего вызова (шторка сверху)
* функция картинка в картинке для просмотра видео из любого места системы
* аудио пресеты

Android 11: habr.com/ru/company/google/blog/506992/

**Основные различия iOS и Android?**

* В первую очередь, очевидно, это разные операционные системы от разных компаний
* Различная целевая аудитория, в т.ч. разный ее размер (У Андроид сегодня ориентировочно 80% всех гаджетов в мире)
* Различная монетизация, в т.ч. ее размер (Согласно статистике, iOS пользуются более платежеспособные люди, которые делают покупки в 3 раза чаще).
* Сложность вхождения в разработку (Для разработки желательно иметь реальный девайс на соответствующей ОС и ПК с соответствующей ОС для разработки. Технику для Android-разработки можно купить значительно дешевле. Даже подписка для разработчика iOS стоит в несколько раз дороже)
* Разное количество разработчиков (Java/Kotlin у Android намного распространеннее и чаще встречается в других сферах, нежели Swift у iOS)
* Отличия в модерации приложений для публикации в магазине приложений (у Android процедура значительно быстрее и проще)
* Отличия с точки зрения дизайна и всего связанного с ним можно узнать, прочитав гайдлайны к каждой системе (Гайдлайны (Guidelines) — набор рекомендаций, правил, принципов от создателей платформы, операционной системы, благодаря которым приложения под эти платформы и ОС от разных разработчиков выглядят единообразно). Вот некоторые из нескольких десятков отличий:
  + Human Interface Guidelines vs Material Design
  + Единицы измерения: pt vs dp
  + Размер экрана: 320 pt x 568 pt vs 360 dp x 640 dp
  + Системный шрифт: San Francisco vs Roboto
  + Android Navigation Bar (в отличие от iOS, у Android есть встроенный инструмент навигации назад)
  + Важность теней в Material (в iOS они почти не используются)
  + Отличия в нейминге в разных местах
  + Отличия в навигации и паттернах (UX)
  + Разные Status Bar
  + Разные Control
  + Разный вид стрелки «Назад» и положение заголовка
  + Action View/Activity View vs Modal Bottom Sheet
  + Разные требования к размеру зоны нажатия

**Виды жестов и т.п.?**

* Скроллинг (scrolling) – прокрутка экрана
* Драг-энд-дроп, дрэг-энд-дроп (drag-and-drop) — «тащить и бросить» — нажать в одном месте, затем двигать и отпустить в другом месте
* Гестуры, жесты (gestures) — разные формы движения мыши, пальца или другого указующего устройства. Могут быть запрограммированы для выполнения определённых действий. Например, движение пальца сверху вниз по экрану мобильного браузера перезагружает страницу
* Тап, тэп (tap) — короткое нажатие пальцем, сродни клику
* Двойной тап, Дабл-тап, дабл-тэп (double tap) — два коротких нажатия пальцем, сродни дабл-клику
* Длинный тап, Тач (touch) — нажатие дольше, чем Тап
* Тач-энд-холд (touch and hold) — нажать и держать
* Флик (flick) — щелчок наискось по экрану в сторону будущего движения изображения экрана, после флика изображение продолжает двигаться по инерции
* Свайп (swipe), Слайд (slide) — продолжительное скольжение пальцем по экрану
* Пинч (pinch) — щипок, сжимающее движение одновременно двумя пальцами по экрану для уменьшения изображения
* Спред / Спрэд (spread), Стретч (stretch: для Microsoft), Пинч-ит-опен (pinch it open: для Apple) — растягивающее движение одновременно двумя пальцами по экрану для увеличения изображения
* Пан, пэн (pan) — медленное движение пальца по экрану для перемещения и разглядывания увеличенной картинки

**Как проверить использование процессора на мобильных устройствах?**

На рынке доступны различные инструменты, такие как Google Play или App Store, из которых можно установить приложения, такие как CPU Monitor, Usemon, CPU Stats, CPU-Z и т. д. Это расширенный инструмент, который записывает информацию о процессах, запущенных на вашем устройстве.

**Объясните критические ошибки, с которыми вы сталкиваетесь при тестировании на мобильных устройствах или в приложениях?**

Объясните пример согласно вашему опыту. Вот топ-10 рисков мобильного приложения.

* Activity monitoring and data retrieval
* Unauthorized dialing, SMS, and payments
* Unauthorized network connectivity (exfiltration or command & control)
* UI Impersonation
* System modification (rootkit, APN proxy config)
* Logic or Time bomb Vulnerabilities
* Sensitive data leakage (inadvertent or side channel)
* Unsafe sensitive data storage
* Unsafe sensitive data transmission
* Hardcoded password/keys

**Что вы знаете о PWA?**

Перспектива разработки мобильного приложения, которое не потребуется скачивать и ждать review из App Store, очень заманчива, ведь аналогов привычного ПО существует несколько: Progressive Web Apps (PWA), Android Instant Apps (AIA) и Accelerated Mobile Pages (AMP). Подробнее можно узнать тут: <https://www.youtube.com/watch?v=i9kmWZGnVEY>  
Или тут: <https://medium.com/effective-developers/how-to-test-pwa-daa1a6eaf7bf>

**Сети и около них**

**Что такое http?**

HTTP — широко распространённый протокол передачи данных, изначально предназначенный для передачи гипертекстовых документов (то есть документов, которые могут содержать ссылки, позволяющие организовать переход к другим документам), сейчас же для любых данных.

Протокол HTTP предполагает использование клиент-серверной структуры передачи данных. Клиентское приложение формирует запрос и отправляет его на сервер, после чего серверное ПО обрабатывает данный запрос, формирует ответ и передаёт его обратно клиенту.

То есть этот протокол не только устанавливает правила обмена информацией, но и служит транспортом для передачи данных — с его помощью браузер загружает содержимое сайта на ваш компьютер или смартфон.

У HTTP есть один недостаток: данные передаются в открытом виде и никак не защищены. На пути из точки А в точку Б информация в интернете проходит через десятки промежуточных узлов, и, если хоть один из них находится под контролем злоумышленника, данные могут перехватить. То же самое может произойти, когда вы пользуетесь незащищённой сетью Wi-Fi, например, в кафе. Для установки безопасного соединения используется протокол HTTPS с поддержкой шифрования.

Защиту данных в HTTPS обеспечивает криптографический протокол SSL/TLS, который шифрует передаваемую информацию. По сути этот протокол является обёрткой для HTTP. Он обеспечивает шифрование данных и делает их недоступными для просмотра посторонними. Протокол SSL/TLS хорош тем, что позволяет двум незнакомым между собой участникам сети установить защищённое соединение через незащищённый канал. При установке безопасного соединения по HTTPS ваш компьютер и сервер сначала выбирают общий секретный ключ, а затем обмениваются информацией, шифруя её с помощью этого ключа. Общий секретный ключ генерируется заново для каждого сеанса связи. Его нельзя перехватить и практически невозможно подобрать — обычно это число длиной более 100 знаков. Этот одноразовый секретный ключ и используется для шифрования всего общения браузера и сервера. Казалось бы, идеальная система, гарантирующая абсолютную безопасность соединения. Однако для полной надёжности ей кое-чего не хватает: гарантии того, что ваш собеседник именно тот, за кого себя выдаёт. Для этой гарантии существует сертификат.

Вам как пользователю сертификат не нужен, но любой сервер (сайт), который хочет установить безопасное соединение с вами, должен его иметь. Сертификат подтверждает две вещи: 1) Лицо, которому он выдан, действительно существует и 2) Оно управляет сервером, который указан в сертификате. Выдачей сертификатов занимаются центры сертификации — что-то вроде паспортных столов. Как и в паспорте, в сертификате содержатся данные о его владельце, в том числе имя (или название организации), а также подпись, удостоверяющая подлинность сертификата. Проверка подлинности сертификата — первое, что делает браузер при установке безопасного HTTPS-соединения. Обмен данными начинается только в том случае, если проверка прошла успешно.

**Компоненты HTTP?**

HTTP определяет следующую структуру запроса (request):

* строка запроса (request line) — определяет тип сообщения
* заголовки запроса (header fields) — характеризуют тело сообщения, параметры передачи и прочие сведения
* тело сообщения (body) — необязательное

HTTP определяет следующую структуру ответного сообщения (response):

* строка состояния (status line), включающая код состояния и сообщение о причине
* поля заголовка ответа (header fields)
* дополнительное тело сообщения (body)

**Методы HTTP-запроса?**

Метод, используемый в HTTP-запросе, указывает, какое действие вы хотите выполнить с этим запросом. Раньше хватало только GET, т.к. считалось, что вы можете хотеть от сервера только получить ответ. Но сейчас вам может понадобиться отредактировать профиль, удалить пост в соц. сети и т.п. Тогда для удобства были созданы различные методы. Вот основные:

* GET: получить подробную информацию о ресурсе
* POST: создать новый ресурс
* PUT: обновить существующий ресурс
* DELETE: Удалить ресурс

Вообще по спецификации HTTP из всех методов сервер должен уметь понимать только GET, а остальные на усмотрение. Но при этом и не задано строго, что сервер должен делать при получении запроса. То есть вы с помощью одного метода можете делать вообще любую операцию. Однако в этом нет никакого практического смысла. В дальнейшем было ввёдено соглашение REST, определившее структуру построения веб-приложений, в том числе и работу с методами.

**Что такое ресурс?**

Ресурс — это ключевая абстракция, на которой концентрируется протокол HTTP. Ресурс — это все, что вы хотите показать внешнему миру через ваше приложение. Например, если мы пишем приложение для управления задачами, экземпляры ресурсов будут следующие:

* Конкретный пользователь
* Конкретная задача
* Список задач

Когда вы разрабатываете RESTful сервисы, вы должны сосредоточить свое внимание на ресурсах приложения. Способ, которым мы идентифицируем ресурс для предоставления, состоит в том, чтобы назначить ему URI — универсальный идентификатор ресурса. Например:

* Создать пользователя: POST /users
* Удалить пользователя: DELETE /users/1
* Получить всех пользователей: GET /users
* Получить одного пользователя: GET /users/1

**Что такое веб-сервис? (WS — Web service)**

За десятилетия существования веба как отрасли сформировались следующие практики межсетевого взаимодействия:

* Обмен файлами по FTP.
* Неструктурированные HTTP-запросы, договорённости между разработчиками.
* Веб-сервисы.
* Экзотика: сокеты, порты, бинарные объекты.

Веб-сервис — это технология, позволяющая системам обмениваться данными друг с другом через сетевое подключение. Обычно веб-сервисы работают поверх протокола HTTP или протокола более высокого уровня. Веб-сервис — просто адрес, ссылка, обращение к которому позволяет получить данные или выполнить действие. То есть это идентифицируемая уникальным веб-адресом (URL-адресом) программная система со стандартизированными интерфейсами, а также HTML-документ сайта, отображаемый браузером пользователя. Веб-службы могут взаимодействовать друг с другом и со сторонними приложениями посредством сообщений, основанных на определённых протоколах (SOAP, XML-RPC и т. д.) и соглашениях (REST). Веб-служба является единицей модульности при использовании сервис-ориентированной архитектуры приложения.

По сути, веб-сервисы — это реализация абсолютно четких интерфейсов обмена данными между различными приложениями, которые написаны не только на разных языках, но и распределены на разных узлах сети.

Именно с появлением веб-сервисов развилась идея SOA — сервис-ориентированной архитектуры веб-приложений (Service Oriented Architecture).

**Отличие сервиса от сервера?**

Сервер может обозначать: - или некую логическую сущность (не обязательно программа), которая выполняет некую задачу (связного набора услуг/сервисов/функций) по отношению к клиенту. - или обычно мощное железо, на котором реализуются какие-то вычислительные функции или хранения данных для пользователей, подключенных удаленно.

**Отличие сервиса от веб-сайта?**

* Веб-сервис не имеет пользовательского интерфейса. Веб-сайт имеет пользовательский интерфейс или графический интерфейс.
* Веб-сервисы предназначены для взаимодействия других приложений через Интернет. Веб-сайты предназначены для использования людьми.
* Веб-сервисы не зависят от платформы, так как используют открытые протоколы. Веб-сайты являются кроссплатформенными, так как требуют настройки для работы в разных браузерах, операционных системах и т. д.
* Доступ к веб-сервисам осуществляется с помощью HTTP-методов — GET, POST, PUT, DELETE и т. д. Доступ к веб-сайтам осуществляется с помощью компонентов GUI - кнопок, текстовых полей, форм и т. д.
* Например, Google maps API — это веб-сервис, который может использоваться веб-сайтами для отображения Карт путем передачи ему координат. Например, ArtOfTesting.com - это веб-сайт, на котором есть коллекция связанных веб-страниц, содержащих учебные пособия.

**Что такое REST, SOAP? В чем отличия?**

* SOAP (Simple Object Access Protocol) — стандартный протокол обмена структурированными сообщениями в распределённой вычислительной среде. Данные передаются в XML.
* REST (Representational State Transfer) — архитектурный стиль взаимодействия компьютерных систем в сети основанный на методах протокола HTTP. Данные по умолчанию передаются в JSON.

SOAP и REST нельзя сравнивать напрямую, поскольку первый — это протокол (или, по крайней мере, пытается им быть), а второй - архитектурный стиль.

Основное различие между SOAP и REST заключается в степени связи между реализациями клиента и сервера. Клиент SOAP работает как пользовательское настольное приложение, тесно связанное с сервером. Между клиентом и сервером существует жесткое соглашение, и ожидается, что все сломается, если какая-либо из сторон что-то изменит. Вам нужно постоянное обновление после любого изменения, но легче определить, выполняется ли контракт.

REST-клиент больше похож на браузер. Это универсальный клиент, который знает, как использовать протокол и стандартизированные методы, и приложение должно соответствовать этому. Вы не нарушаете стандарты протокола, создавая дополнительные методы, вы используете стандартные методы и создаете с ними действия для своего типа медиа. Если все сделано правильно, связности будет меньше, и с изменениями можно справиться более изящно.

То есть SOAP более применим в сложных архитектурах, где взаимодействие с объектами выходит за рамки теории CRUD, а вот в тех приложениях, которые не покидают рамки данной теории, вполне применимым может оказаться именно REST ввиду своей простоты и прозрачности. Действительно, если любым объектам вашего сервиса не нужны более сложные взаимоотношения, кроме: «Создать», «Прочитать», «Изменить», «Удалить» (как правило — в 99% случаев этого достаточно), возможно, именно REST станет правильным выбором. Кроме того, REST по сравнению с SOAP, может оказаться и более производительным, так как не требует затрат на разбор сложных XML команд на сервере (выполняются обычные HTTP запросы — PUT, GET, POST, DELETE). Хотя SOAP, в свою очередь, более надежен и безопасен.

**Что такое JSON, XML?**

JSON - текстовый формат обмена данными, основанный на JavaScript (но он не зависит от языка).

XML — это язык разметки. Является выбором по умолчанию для обмена данными, остается легко читаемым, даже при больших массивах информации.

JSON благодаря популярности технологии API REST, получил импульс развития в программировании API и веб-сервисов. Это текстовый, легкий и простой в разборе формат данных, не требующий дополнительного кода для анализа. Таким образом, JSON помогает ускорить обмен данными и для веб-сервисов, которые должны просто возвращать много данных и отображать их.

Пример JSON:

[

{

«title»:«bananas»,

«count»:«1000»,

«description»:[«500 green»,«500 yellow»]

},

{

…

}

]

В python аналогичная структура данных – словари. То есть это просто набор ключ: значение. При этом ключ должен быть уникальным, значений может быть любое количество. Допускается вложенность (значением может быть другой json или список).

Пример XML:

<company>

<company-id>12345</company-id>

<name lang="ru">Абракадабра</name>

<country lang="ru">Россия</country>

<phone>

<number>+7 (999) 999-99-99</number>

<ext>777</ext>

<info>Приёмная</info>

<type>phone</type>

</phone>

</company>

Как видно, XML похож на HTML, однако здесь теги не предопределены.

**Коды ответов/состояния сервера с примерами? (HTTP status code)**

Несколько из них могут спросить чуть конкретнее, чем просто название, обычно на Ваш же выбор.

Иногда на собеседовании можно услышать вопрос: «Что дают эти коды ответа и что с ними можно делать?». На него настолько обширный ответ, что в рамках данной статьи это было бы не уместно, но конкретно для тестировщика чаще всего это просто удобное понимание, как именно отреагировал сервер на web или API запрос.

* Информационные (100-105)
* Успешные (200-226)
* Перенаправление (300-307)
* Ошибка клиента (400-499)
* Ошибка сервера (500-510)
* \*Коды Microsoft Server (600-955) (коды ошибок, возникающих при создании коммутируемых подключений или VPN-подключений на клиентских компьютерах под управлением Windows 2000, Windows XP и Windows Server 2003. Примечание. Ошибки с кодами больше 900 появляются только при подключении к серверу маршрутизации и удаленного доступа под управлением Windows 2000 или операционной системы более поздней версии.)

|  |  |  |
| --- | --- | --- |
| Код ответа | Название | Описание |
| 100 | Continue | «Продолжить». Этот промежуточный ответ указывает, что запрос успешно принят и клиент может продолжать присылать запросы либо проигнорировать этот ответ, если запрос был завершён. |
| 101 | Switching Protocol | «Переключение протокола». Этот код присылается в ответ на запрос клиента, содержащий заголовок Upgrade, и указывает, что сервер переключился на протокол, который был указан в заголовке. Эта возможность позволяет перейти на несовместимую версию протокола и обычно не используется. |
| 102 | Processing | «В обработке». Этот код указывает, что сервер получил запрос и обрабатывает его, но обработка еще не завершена. |
| 103 | Early Hints | «Ранние подсказки». В ответе сообщаются ресурсы, которые могут быть загружены заранее, пока сервер будет подготовливать основной ответ. |
| 200 | OK | «Успешно». Запрос успешно обработан. Что значит «успешно», зависит от метода HTTP, который был запрошен:GET: «ПОЛУЧИТЬ». Запрошенный ресурс был найден и передан в теле ответа.HEAD: «ЗАГОЛОВОК». Заголовки переданы в ответе.POST: «ПОСЫЛКА». Ресурс, описывающий результат действия сервера на запрос, передан в теле ответа.TRACE: «ОТСЛЕЖИВАТЬ». Тело ответа содержит тело запроса полученного сервером. |
| 201 | Created | «Создано». Запрос успешно выполнен и в результате был создан ресурс. Этот код обычно присылается в ответ на запрос PUT «ПОМЕСТИТЬ». |
| 202 | Accepted | «Принято». Запрос принят, но ещё не обработан. Не поддерживаемо, т.е., нет способа с помощью HTTP отправить асинхронный ответ позже, который будет показывать итог обработки запроса. Это предназначено для случаев, когда запрос обрабатывается другим процессом или сервером, либо для пакетной обработки. |
| 203 | Non-Authoritative Information | «Информация не авторитетна». Этот код ответа означает, что информация, которая возвращена, была предоставлена не от исходного сервера, а из какого-нибудь другого источника. Во всех остальных ситуациях более предпочтителен код ответа 200 OK. |
| 204 | No Content | «Нет содержимого». Нет содержимого для ответа на запрос, но заголовки ответа, которые могут быть полезны, присылаются. Клиент может использовать их для обновления кешированных заголовков полученных ранее для этого ресурса. |
| 205 | Reset Content | «Сбросить содержимое». Этот код присылается, когда запрос обработан, чтобы сообщить клиенту, что необходимо сбросить отображение документа, который прислал этот запрос. |
| 206 | Partial Content | «Частичное содержимое». Этот код ответа используется, когда клиент присылает заголовок диапазона, чтобы выполнить загрузку отдельно, в несколько потоков. |
| 300 | Multiple Choice | «Множественный выбор». Этот код ответа присылается, когда запрос имеет более чем один из возможных ответов. И User-agent или пользователь должен выбрать один из ответов. Не существует стандартизированного способа выбора одного из полученных ответов. |
| 301 | Moved Permanently | «Перемещён на постоянной основе». Этот код ответа значит, что URI запрашиваемого ресурса был изменен. Возможно, новый URI будет предоставлен в ответе. |
| 302 | Found | «Найдено». Этот код ответа значит, что запрошенный ресурс временно изменен. Новые изменения в URI могут быть доступны в будущем. Таким образом, этот URI, должен быть использован клиентом в будущих запросах. |
| 303 | See Other | «Просмотр других ресурсов». Этот код ответа присылается, чтобы направлять клиента для получения запрашиваемого ресурса в другой URI с запросом GET. |
| 304 | Not Modified | «Не модифицировано». Используется для кэширования. Это код ответа значит, что запрошенный ресурс не был изменен. Таким образом, клиент может продолжать использовать кэшированную версию ответа. |
| 305 | Use Proxy | «Использовать прокси». Это означает, что запрошенный ресурс должен быть доступен через прокси. Этот код ответа в основном не поддерживается из соображений безопасности. |
| 306 | Switch Proxy | Больше не использовать. Изначально подразумевалось, что « последующие запросы должны использовать указанный прокси.» |
| 307 | Temporary Redirect | «Временное перенаправление». Сервер отправил этот ответ, чтобы клиент получил запрошенный ресурс на другой URL-адрес с тем же методом, который использовал предыдущий запрос. Данный код имеет ту же семантику, что код ответа 302 Found, за исключением того, что агент пользователя не должен изменять используемый метод HTTP: если в первом запросе использовался POST, то во втором запросе также должен использоваться POST. |
| 308 | Permanent Redirect | «Перенаправление на постоянной основе». Это означает, что ресурс теперь постоянно находится в другом URI, указанном в заголовке Location: HTTP Response. Данный код ответа имеет ту же семантику, что и код ответа 301 Moved Permanently, за исключением того, что агент пользователя не должен изменять используемый метод HTTP: если POST использовался в первом запросе, POST должен использоваться и во втором запросе.Примечание: Это экспериментальный код ответа, Спецификация которого в настоящее время находится в черновом виде. |
| 400 | Bad Request | «Плохой запрос». Этот ответ означает, что сервер не понимает запрос из-за неверного синтаксиса. |
| 401 | Unauthorized | «Неавторизовано». Для получения запрашиваемого ответа нужна аутентификация. Статус похож на статус 403, но, в этом случае, аутентификация возможна. |
| 402 | Payment Required | «Необходима оплата». Этот код ответа зарезервирован для будущего использования. Первоначальная цель для создания этого когда была в использовании его для цифровых платежных систем(на данный момент не используется). |
| 403 | Forbidden | «Запрещено». У клиента нет прав доступа к содержимому, поэтому сервер отказывается дать надлежащий ответ. |
| 404 | Not Found | «Не найден». Сервер не может найти запрашиваемый ресурс. Код этого ответа, наверно, самый известный из-за частоты его появления в вебе. |
| 405 | Method Not Allowed | «Метод не разрешен». Сервер знает о запрашиваемом методе, но он был деактивирован и не может быть использован. Два обязательных метода, GET и HEAD, никогда не должны быть деактивированы и не должны возвращать этот код ошибки. |
| 406 | Not Acceptable | Этот ответ отсылается, когда веб сервер после выполнения server-driven content negotiation, не нашел контента, отвечающего критериям, полученным из user agent. |
| 407 | Proxy Authentication Required | Этот код ответа аналогичен коду 401, только аутентификация требуется для прокси сервера. |
| 408 | Request Timeout | Ответ с таким кодом может прийти, даже без предшествующего запроса. Он означает, что сервер хотел бы отключить это неиспользуемое соеднинение. Этот метод используется все чаще с тех пор, как некоторые браузеры, вроде Chrome и IE9, стали использовать HTTP механизмы предварительного соединения для ускорения серфинга (смотрите баг 634278, будущей реализации этого механизма в Firefox). Также учитывайте, что некоторые серверы прерывают соединения не отправляя подобных сообщений. |
| 409 | Conflict | Этот ответ отсылается, когда запрос конфликтует с текущим состоянием сервера. |
| 410 | Gone | Этот ответ отсылается, когда запрашиваемый контент удален с сервера. |
| 411 | Length Required | Запрос отклонен, потому что сервер требует указание заголовка Content-Length, но он не указан. |
| 412 | Precondition Failed | Клиент указал в своих заголовках условия, которые сервер не может выполнить |
| 413 | Request Entity Too Large | Размер запроса превышает лимит, объявленный сервером. Сервер может закрыть соединение, вернув заголовок Retry-After |
| 414 | Request-URI Too Long | URI запрашиваемый клиентом слишком длинный для того, чтобы сервер смог его обработать |
| 415 | Unsupported Media Type | Медиа формат запрашиваемых данных не поддерживается сервером, поэтому запрос отклонен |
| 416 | Requested Range Not Satisfiable | Диапазон указанный заголовком запроса Range не может быть выполнен; возможно, он выходит за пределы переданного URI |
| 417 | Expectation Failed | Этот код ответа означает, что ожидание, полученное из заголовка запроса Expect, не может быть выполнено сервером. |
| 500 | Internal Server Error | «Внутренняя ошибка сервера». Сервер столкнулся с ситуацией, которую он не знает, как обработать. |
| 501 | Not Implemented | «Не выполнено». Метод запроса не поддерживается сервером и не может быть обработан. Единственные методы, которые сервера должны поддерживать (и, соответственно, не должны возвращать этот код) - GET и HEAD. |
| 502 | Bad Gateway | «Плохой шлюз». Эта ошибка означает что сервер, во время работы в качестве шлюза для получения ответа, нужного для обработки запроса, получил недействительный (недопустимый) ответ. |
| 503 | Service Unavailable | «Сервис недоступен». Сервер не готов обрабатывать запрос. Зачастую причинами являются отключение сервера или то, что он перегружен. Обратите внимание, что вместе с этим ответом удобная для пользователей(user-friendly) страница должна отправлять объяснение проблемы. Этот ответ должен использоваться для временных условий и Retry-After: HTTP-заголовок должен, если возможно, содержать предполагаемое время до восстановления сервиса. Веб-мастер также должен позаботиться о заголовках, связанных с кэшем, которые отправляются вместе с этим ответом, так как эти ответы, связанные с временными условиями, обычно не должны кэшироваться. |
| 504 | Gateway Timeout | Этот ответ об ошибке предоставляется, когда сервер действует как шлюз и не может получить ответ вовремя. |
| 505 | HTTP Version Not Supported | «HTTP-версия не поддерживается». HTTP-версия, используемая в запроcе, не поддерживается сервером. |

**Почему ошибка 404 относится к 4\*\* — клиентской, если по идее должна быть 5\*\*?**

Хотя интуитивно можно подумать, что данная ошибка должна относиться к ошибкам со стороны сервера, 404 по задумке является клиентской ошибкой, то есть подразумевается, что клиент (Вы) должен был знать, что URL страницы был перемещен или удален и Вы пытаетесь открыть несуществующую страницу.

**Какие еще бывают протоколы?**

* [FTP](https://ru.wikipedia.org/wiki/FTP) (File Transfer Protocol) — это протокол передачи файлов со специального файлового сервера на компьютер пользователя. FTP дает возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удаленным компьютером, пользователь может скопировать файл с удаленного компьютера на свой или скопировать файл со своего компьютера на удаленный.
* [POP3](https://ru.wikipedia.org/wiki/POP3) (Post Office Protocol) — это стандартный протокол почтового соединения. Серверы POP обрабатывают входящую почту, а протокол POP предназначен для обработки запросов на получение почты от клиентских почтовых программ.
* [SMTP](https://ru.wikipedia.org/wiki/SMTP) (Simple Mail Transfer Protocol) — протокол, который задает набор правил для передачи почты. Сервер SMTP возвращает либо подтверждение о приеме, либо сообщение об ошибке, либо запрашивает дополнительную информацию.
* [TELNET](https://ru.wikipedia.org/wiki/TELNET) — это протокол удаленного доступа. TELNET дает возможность абоненту работать на любой ЭВМ находящейся с ним в одной сети, как на своей собственной, то есть запускать программы, менять режим работы и так далее. На практике возможности ограничиваются тем уровнем доступа, который задан администратором удаленной машины.
* TCP — сетевой протокол, отвечающий за передачу данных в сети Интернет.
* UDP — это тоже транспортный протокол передачи данных, но без подтверждения доставки
* Ethernet — протокол, определяющий стандарты сети на физическом и канальном уровнях.

**TCP/IP это?**

TCP/IP — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет.

Набор интернет-протоколов — это концептуальная модель и набор коммуникационных протоколов, используемых в Интернете и подобных компьютерных сетях. Он широко известен как TCP/IP, поскольку базовые протоколы в пакете — это протокол управления передачей (TCP) и интернет-протокол (IP).

Набор интернет-протоколов обеспечивает сквозную передачу данных, определяющую, как данные должны пакетироваться, обрабатываться, передаваться, маршрутизироваться и приниматься. Эта функциональность организована в четыре слоя абстракции, которые классифицируют все связанные протоколы в соответствии с объемом задействованных сетей.

От самого низкого до самого высокого уровня:

* уровень связи, содержащий методы связи для данных, которые остаются в пределах одного сегмента сети;
* интернет-уровень, обеспечивающий межсетевое взаимодействие между независимыми сетями;
* транспортный уровень, обрабатывающий связь между хостами;
* прикладной уровень, который обеспечивает обмен данными между процессами для приложений.

**Что такое куки (cookies)?**

Файл cookie HTTP (файл cookie Интернета, файл cookie браузера) представляет собой небольшой фрагмент данных (часть http заголовка), который сервер отправляет в веб-браузер пользователя. Браузер может сохранить его и отправить обратно с последующими запросами на тот же сервер. Как правило, он используется для определения того, поступили ли два запроса от одного и того же браузера — например, для входа пользователя в систему. Он запоминает информацию stateful для stateless протокола HTTP. Куки в основном используются для трех целей:

* Управление сессиями: Логины, корзины покупок, результаты игр и все, что сервер должен запомнить
* Пользовательские настройки, темы и другие настройки
* Запись и анализ поведения пользователя

Виды кук:

* Сессионные cookie, также известные как временные cookie, существуют только во временной памяти, пока пользователь находится на странице веб-сайта. Браузеры обычно удаляют сессионные cookie после того, как пользователь закрывает окно браузер. В отличие от других типов cookie, сессионные cookie не имеют истечения срока действия, и поэтому браузеры понимают их как сессионные.
* Вместо того, чтобы удаляться после закрытия браузера, как это делают сессионные cookie, постоянные cookie-файлы удаляются в определённую дату или через определённый промежуток времени. Это означает, что информация о cookie будет передаваться на сервер каждый раз, когда пользователь посещает веб-сайт, которому эти cookie принадлежат. По этой причине постоянные cookie иногда называются следящие cookie, поскольку они могут использоваться рекламодателями для записи о предпочтениях пользователя в течение длительного периода времени. Однако, они также могут использоваться и в «мирных» целях, например, чтобы избежать повторного ввода данных при каждом посещении сайта.
* Обычно атрибут домена cookie совпадает с доменом, который отображается в адресной строке веб-браузера. Это называется первый файл cookie. Однако сторонний файл cookie принадлежит домену, отличному от того, который указан в адресной строке. Этот тип файлов cookie обычно появляется, когда веб-страницы содержат контент с внешних веб-сайтов, например, рекламные баннеры. Это открывает возможности для отслеживания истории посещений пользователя и часто используется рекламодателями для предоставления релевантной рекламы каждому пользователю.
* Супер-cookie — это cookie-файл с источником домена верхнего уровня (например, .ru) или общедоступным суффиксом (например, .co.uk). Обычные cookie, напротив, имеют происхождение от конкретного доменного имени, например, example.com.

Супер-cookie могут быть потенциальной проблемой безопасности и поэтому часто блокируются веб-браузерами. Если браузер разблокирует вредоносный веб-сайт, злоумышленник может установить супер-cookie и потенциально нарушить или выдать себя за законные запросы пользователей на другой веб-сайт, который использует тот же домен верхнего уровня или общедоступный суффикс, что и вредоносный веб-сайт. Например, супер-cookie с происхождением .com может злонамеренно повлиять на запрос к example.com, даже если файл cookie не был создан с сайта example.com. Это может быть использовано для подделки логинов или изменения информации пользователя.

* Поскольку cookie можно очень легко удалить из браузера, программисты ищут способы идентифицировать пользователей даже после полной очистки истории браузера. Одним из таких решений являются зомби-cookie (или evercookie, или persistent cookie) — не удаляемые или трудно удаляемые cookie, которые можно восстановить в браузере с помощью JavaScript. Это возможно потому, что для хранения куков сайт одновременно использует все доступные хранилища браузера (HTTP ETag, Session Storage, Local Storage, Indexed DB), в том числе и хранилища приложений, таких как Flash Player (Local Shared Objects), Microsoft Silverlight (Isolated Storage) и Java (Java persistence API). Когда программа обнаруживает отсутствие в браузере cookie-файла, информация о котором присутствует в других хранилищах — она тут же восстанавливает его на место и, тем самым, идентифицирует пользователя для сайта.

**Разница между cookie и сессией/сеансом?**

* Cookies — это файлы на стороне клиента, которые содержат информацию о пользователе. Сессия - это файлы на стороне сервера, которые содержат информацию о пользователе.
* Cookie-файл заканчивается в зависимости от срока, установленного для него. Сессия заканчивается, когда пользователь закрывает свой браузер.
* Вам не нужно запускать cookie, так как он хранится на вашем локальном компьютере. В PHP перед использованием $ \_SESSION вы должны написать session\_start (); Аналогично для других языков.
* Официальный максимальный размер файла cookie составляет 4 КБ. В течение сессии вы можете хранить столько данных, сколько захотите. Единственное ограничение, которое вы можете достичь — это максимальный объем памяти, который скрипт может использовать за один раз, по умолчанию 128 МБ.
* Файл cookie не зависит от сессии. Сессия зависит от Cookie.

**Отличие stateless и stateful?**

stateful — модель, при которой объект содержит информацию о своем состоянии, все методы работают в контексте его состояния

stateless не предоставляют эту информацию. Все методы объекта работают вне какого-либо контекста или локального состояния объекта, которого в этом случае просто нет. Не делается предположений о состоянии сессии, все изменения атомарны, нет каких-то сессионных переменных на сервере, помнящих результат предыдущего запроса. Они каждый раз дают один и тот же неизменный ответ на один и тот же запрос, функцию или вызов метода. HTTP не имеет состояния в необработанном виде — если вы выполняете GET для определенного URL, вы получаете (теоретически) один и тот же ответ каждый раз.

**Различия методов GET и POST?**

Основное состоит в способе передачи данных веб-формы обрабатывающему скрипту, а именно:

* Метод GET отправляет скрипту всю собранную информацию формы как часть URL: <http://www.komtet.ru/script.php?login=admin&name=komtet>
* Метод POST передает данные таким образом, что пользователь сайта уже не видит передаваемые скрипту данные: <http://www.komtet.ru/script.php>

Оба метода успешно передают необходимую информацию из веб-формы скрипту, поэтому при выборе того или иного метода, который будет наиболее подходить сайту, нужно учитывать следующие факторы:

* Принцип работы метода GET ограничивает объём передаваемой скрипту информации;
* Так как метод GET отправляет скрипту всю собранную информацию формы как часть URL (то есть в открытом виде), то это может пагубно повлиять на безопасность сайта;
* Страницу, сгенерированную методом GET, можно пометить закладкой (адрес страницы будет всегда уникальный), а страницу, сгенерированную метод POST нельзя (адрес страницы остается неизменным, так как данные в URL не подставляются);
* Используя метод GET можно передавать данные не через веб-форму, а через URL страницы, введя необходимые значения через знак &: <http://www.komtet.ru/script.php?login=admin&name=komtet>
* Метод POST в отличие от метода GET позволяет передавать запросу файлы;
* При использовании метода GET существует риск того, что поисковый робот может выполнить тот или иной открытый запрос.

**Клиент — серверная архитектура?**

* Сервер – логический процесс, который обеспечивает некоторый сервис по запросу от клиента. Обычно сервер не только выполняет запрос, но и управляет очередностью запросов, буферами обмена, извещает своих клиентов о выполнении запроса и т. д.
* Клиент – процесс, который запрашивает обслуживание от сервера. Процесс не является клиентом по каким-то параметрам своей структуры, он является клиентом только по отношению к серверу.
* Сеть, протоколы – третий компонент, который обеспечивает обмен информацией между клиентом и сервером.

При взаимодействии клиента и сервера инициатором диалога с сервером, как правило, является клиент, сервер сам не инициирует совместную работу.

Технология клиент-сервер — шаблон проектирования, основа для создания веб-приложений, взаимодействие, при котором одна программа (клиент) запрашивает услугу (выполнение какойлибо совокупности действий), а другая программа (сервер) ее выполняет.

Двухзвенная архитектура клиент-сервер:

* «Толстый клиент»: на сервере реализованы главным образом функции доступа к базам данных, а основные прикладные вычисления выполняются на стороне клиента.
* «Тонкий клиент»: на сервере выполняется основная часть прикладной обработки данных, а на клиентские рабочие станции передаются уже результаты обработки данных для просмотра и анализа пользователем с возможностью их последующей обработки (в минимальном объёме).

Многоуровневая архитектура «клиент-сервер»:

Разновидность архитектуры клиент-сервер, в которой функция обработки данных вынесена на один или несколько отдельных серверов. Это позволяет разделить функции хранения, обработки и представления данных для более эффективного использования возможностей серверов и клиентов.

**Уровни OSI?**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель OSI | | | | |
| Уровень (layer) | | Тип данных (PDU) | Функции | Примеры |
| Host layers | 7. Прикладной (application) | Данные | Доступ к сетевым службам | HTTP, FTP, POP3, WebSocket |
| 6. Представления (presentation) | Представление и шифрование данных | ASCII, EBCDIC |  |  |
| 5. Сеансовый (session) | Управление сеансом связи | RPC, PAP, L2TP |  |  |
| 4. Транспортный (transport) | Сегменты(segment) /Дейтаграммы (datagram) | Прямая связь между конечными пунктами и надёжность | TCP, UDP, SCTP, PORTS |  |
| Media layers | 3. Сетевой (network) | Пакеты (packet) | Определение маршрута и логическая адресация | IPv4, IPv6, IPsec, AppleTalk |
| 2. Канальный (data link) | Биты (bit)/ Кадры (frame) | Физическая адресация | PPP, IEEE 802.22, Ethernet, DSL, ARP, сетевая карта. |  |
| 1. Физический (physical) | Биты (bit) | Работа со средой передачи, сигналами и двоичными данными | USB, кабель («витая пара», коаксиальный, оптоволоконный), радиоканал |  |

Вместо OSI в реальности актуальнее знать TCP/IP. Тестировщики и разработчики почти всегда работают на прикладном уровне. Ниже может быть только тестирование VoIP и т.п.

**Что вы подразумеваете под потоковыми медиа? (Streaming media)**

Потоковая передача — это процесс загрузки данных с сервера.

Потоковое мультимедиа - мультимедиа, которое непрерывно получает пользователь от провайдера потокового вещания. Это понятие применимо как к информации, распространяемой через телекоммуникации, так и к информации, которая изначально распространялась посредством потокового вещания (например, радио, телевидение)

**Основные команды Linux?**

* pwd - когда вы впервые открываете терминал, вы попадаете в домашний каталог вашего пользователя. Чтобы узнать, в каком каталоге вы находитесь, вы можете использовать команду «pwd». Это команда выводит полный путь от корневого каталога к текущему рабочему каталогу: в контексте которого (по умолчанию) будут исполняться вводимые команды. Корень является основой файловой системы Linux. Обозначается косой чертой (/). Каталог пользователя обычно выглядит как «/ home / username».
* ls - используйте команду «ls», чтобы узнать, какие файлы находятся в каталоге, в котором вы находитесь. Вы можете увидеть все скрытые файлы, используя команду «ls -a».
* cd - используйте команду «cd», чтобы перейти в каталог. Например, если вы находитесь в домашней папке и хотите перейти в папку загрузок, вы можете ввести «cd Downloads». Помните, что эта команда чувствительна к регистру, и вы должны ввести имя папки в точности так, как оно есть. Но есть один нюанс. Представьте, что у вас есть папка с именем «Raspberry Pi». В этом случае, когда вы вводите «cd Raspberry Pi», оболочка примет второй аргумент команды как другой, поэтому вы получите сообщение об ошибке, говорящее о том, что каталог не существует. Здесь вы можете использовать обратную косую черту, то есть: «cd Raspberry/ Pi». Пробелы работают так: если вы просто наберете «cd» и нажмете клавишу ввода, вы попадете в домашний каталог. Чтобы вернуться из папки в папку до этого, вы можете набрать «cd ..». Две точки возвращают в предыдущий каталог.
* mkdir и rmdir - используйте команду mkdir, когда вам нужно создать папку или каталог. Например, если вы хотите создать каталог под названием «DIY», вы можете ввести «mkdir DIY». Помните, как уже было сказано, если вы хотите создать каталог с именем «DIY Hacking», вы можете ввести «mkdir DIY/ Hacking». Используйте rmdir для удаления каталога. Но rmdir можно использовать только для удаления пустой директории. Чтобы удалить каталог, содержащий файлы, используйте команду rm.
* rm - используйте команду rm для удаления файлов и каталогов. Используйте «rm -r», чтобы удалить только каталог. Он удаляет как папку, так и содержащиеся в ней файлы при использовании только команды rm.
* touch - команда touch используется для создания файла. Это может быть что угодно, от пустого txt-файла до пустого zip-файла. Например, «touch new.txt».
* man и --help - Чтобы узнать больше о команде и о том, как ее использовать, используйте команду man. Показывает справочные страницы команды. Например, «man ls» показывает справочные страницы команды ls. Ввод имени команды и аргумента помогает показать, каким образом можно использовать команду (например, cd --help).
* cp - используйте команду cp для копирования файлов через командную строку. Он принимает два аргумента: первый — это местоположение файла, который нужно скопировать, второй - куда копировать.
* mv - используйте команду mv для перемещения файлов через командную строку. Мы также можем использовать команду mv для переименования файла. Например, если мы хотим переименовать файл «text» в «new», мы можем использовать «mv text new». Он принимает два аргумента, как и команда cp.
* locate - команда locate используется для поиска файла в системе Linux, так же, как команда поиска в Windows. Эта команда полезна, когда вы не знаете, где файл сохранен или фактическое имя файла. Использование аргумента -i с командой помогает игнорировать регистр (не имеет значения, является ли он прописным или строчным). Итак, если вам нужен файл со словом «hello», он дает список всех файлов в вашей системе Linux, содержащих слово «hello», когда вы вводите «locate -i hello». Если вы помните два слова, вы можете разделить их звездочкой (\*). Например, чтобы найти файл, содержащий слова «hello» и «this», вы можете использовать команду «locate -i \* hello \* this».

Промежуточные команды:

* echo - команда «echo» помогает нам перемещать некоторые данные, обычно текст, в файл. Например, если вы хотите создать новый текстовый файл или добавить в уже созданный текстовый файл, вам просто нужно ввести «echo hello, меня зовут hich >> new.txt». Вам не нужно разделять пробелы с помощью обратной косой черты здесь, потому что мы заключаем в две треугольные скобки, когда мы заканчиваем то, что нам нужно написать.
* cat - Используйте команду cat для отображения содержимого файла. Обычно используется для удобного просмотра программ.
* nano, vi, jed - nano и vi уже установлены текстовые редакторы в командной строке Linux. Команда nano — хороший текстовый редактор, который помечает ключевые слова цветом и может распознавать большинство языков. И vi проще, чем nano. Вы можете создать новый файл или изменить файл с помощью этого редактора. Например, если вам нужно создать новый файл с именем «check.txt», вы можете создать его с помощью команды «nano check.txt». Вы можете сохранить ваши файлы после редактирования, используя последовательность Ctrl + X, затем Y (или N для no). По моему опыту, использование nano для редактирования HTML выглядит не очень хорошо из-за его цвета, поэтому я рекомендую jed текстовый редактор. Мы скоро приступим к установке пакетов.
* sudo - широко используемая команда в командной строке Linux, sudo означает «SuperUser Do». Поэтому, если вы хотите, чтобы любая команда выполнялась с правами администратора или root, вы можете использовать команду sudo. Например, если вы хотите отредактировать файл, такой как viz. alsa-base.conf, для которого требуются права root, вы можете использовать команду — sudo nano alsa-base.conf. Вы можете ввести корневую командную строку с помощью команды «sudo bash», а затем ввести свой пароль пользователя. Вы также можете использовать команду «su», но перед этим вам нужно установить пароль root. Для этого вы можете использовать команду «sudo passwd» (не с орфографической ошибкой, это passwd). Затем введите новый пароль root.
* df - используйте команду df, чтобы увидеть доступное дисковое пространство в каждом из разделов вашей системы. Вы можете просто ввести df в командной строке и увидеть каждый смонтированный раздел и его использованное / доступное пространство в % и в килобайтах. Если вы хотите, чтобы оно отображалось в мегабайтах, вы можете использовать команду «df -m».
* du - Используйте du, чтобы узнать, как файл используется в вашей системе. Если вы хотите узнать размер занимаемого места на диске для конкретной папки или файла в Linux, вы можете ввести команду df и имя папки или файла. Например, если вы хотите узнать размер дискового пространства, используемое папкой документов в Linux, вы можете использовать команду «du Documents». Вы также можете использовать команду «ls -lah», чтобы просмотреть размеры всех файлов в папке.
* tar - Используйте tar для работы с tarballs (или файлами, сжатыми в архиве tarball) в командной строке Linux. У него длинный список применений. Он может использоваться для сжатия и распаковки различных типов архивов tar, таких как .tar, .tar.gz, .tar.bz2 и т. д. Это работает на основе аргументов, данных ему. К примеру, «tar -cvf» для создания .tar архива, -xvf для распаковки .tar архива, -tvf для просмотра содержимого архива и т. д.
* zip, unzip - используйте zip для сжатия файлов в zip-архив и unzip для извлечения файлов из zip-архива.
* uname - используйте uname, чтобы показать информацию о системе, в которой работает ваш дистрибутив Linux. Использование команды «uname -a» выводит большую часть информации о системе: дату выпуска ядра, версию, тип процессора и т. д.
* apt-get - используйте apt для работы с пакетами в командной строке Linux. Используйте apt-get для установки пакетов. Это команда требует прав суперпользователя, поэтому используйте команду sudo с ним. Например, если вы хотите установить текстовый редактор jed (как я упоминал ранее), мы можем ввести команду «sudo apt-get install jed». Точно так же любые пакеты могут быть установлены следующим образом. Рекомендуется обновлять ваш репозиторий каждый раз, когда вы пытаетесь установить новый пакет. Вы можете сделать это, набрав «sudo apt-get update». Вы можете обновить систему, набрав «sudo apt-get upgrade». Мы также можем обновить дистрибутив, набрав «sudo apt-get dist-upgrade». Команда «apt-cache search» используется для поиска пакета. Если вы хотите найти его, вы можете ввести «apt-cache search jed» (для этого не требуется root).
* chmod - используйте chmod, чтобы сделать файл исполняемым и изменить разрешения, предоставленные ему в Linux. Представьте, что на вашем компьютере есть код Python с именем numbers.py. Вам нужно будет запускать «python numbers.py» каждый раз, когда вам нужно его запустить. Вместо этого, когда вы делаете его исполняемым, вам просто нужно запустить «numbers.py» в терминале, чтобы запустить файл. Чтобы сделать файл исполняемым, вы можете использовать команду «chmod + x numbers.py» в этом случае. Вы можете использовать «chmod 755 numbers.py», чтобы дать ему права root, или «sudo chmod + x numbers.py» для исполняемого файла root.
* hostname - Используйте команду hostname, чтобы узнать ваше имя в вашем хосте или сети. По сути, он отображает ваше имя хоста и IP-адрес. Просто набрав «hostname», вы получите имя хоста. Набрав «hostname -I», вы получите свой IP-адрес в сети.
* ping - используйте ping для проверки вашего соединения с сервером. Википедия говорит: «Ping — это утилита для администрирования компьютерной сети, используемая для проверки доступности хоста в сети Интернет-протокола (IP)». Например, когда вы набираете, , «ping [google.com](http://google.com/)», он проверяет, может ли он подключиться к серверу и вернуться обратно. Он измеряет это время в оба конца и дает вам подробную информацию о нем. Использовать эту команду можно и для проверки интернет-соединения. Если он пингует сервер Google (в данном случае) — интернет-соединение активно!

**Почему важно тестировать в разных браузерах?**

Приложения и сайты в разных браузерах могут вести себя по-разному. Это связано с тем, что любой из браузеров имеет собственные движки, надстройки, плагины, а также различия в десктопной и мобильной версиях. Кроссбраузерное тестирование призвано сгладить эти различия, сделав разработку более или менее универсальной.

Почему возникают кросс-браузерные ошибки:

* Иногда сами браузеры содержат баги или по-разному внедряют функции. Часто это происходит из-за попытки заполучить конкурентное преимущество.
* Некоторые браузеры могут иметь разные уровни поддержки технологических функций для других браузеров. JavaScript фичи, скорее всего, не будут работать на старых браузерах.
* Некоторые девайсы могут содержать ограничения, которые могут заставить веб-сайт работать медленнее или некорректно отображаться. Например, если сайт был спроектирован под десктоп, то вполне вероятно, что его контент будет сложно читать на мобильном устройстве.
* Приступать к тестированию сайта в популярных браузерах следует уже после того как он проверен на дефекты другими видами тестирования. Только в этом случае можно будет сказать, что выявленные некорректные сценарии имеют отношение именно к особенностям браузера, а не были пропущены на других стадиях. Разумеется, при этом ошибка должна проявляться не во всех браузерах. Внимание нужно также уделить сочетанию операционной системы и браузера, выбрав наиболее распространенные из них.

**Адаптивный веб-дизайн vs. Отзывчивый веб-дизайн, в чем разница? (Adaptive Vs. Responsive)**

Responsive Design (RWD) — отзывчивый дизайн — проектирование сайта с определенными значениями свойств, например, гибкая сетка макета, которые позволяют одному макету работать на разных устройствах;

Помимо своей изменяющейся структуры, у респонсив дизайна есть несколько других преимуществ:

1. Одинаковый внешний вид ресурса в разных браузерах и на различных платформах
2. Наличие у сайта одинакового URL, что способствует SEO-оптимизации
3. Разработчикам необходимо обслуживать лишь один сайт, что позволяет сократить время, затрачиваемое на дизайн и контент

И хотя положительные стороны респонсивного дизайна очевидны, у этого метода существует ряд недостатков. Самым большим из них является скорость загрузки, которая значительно снижается из-за высокого разрешения изображений и других визуальных элементов, необходимых для оформления внешнего вида ресурса.

Adaptive Design (AWD) — адаптивный дизайн, или динамический показ — проектирование сайта с условиями, которые изменяются в зависимости от устройства, базируясь на нескольких макетах фиксированной ширины.

Для создания отзывчивых макетов используются медиазапросы и относительные размеры элементов сетки, заданные с помощью %. В адаптивном дизайне серверные скрипты сначала определяют тип устройства, с помощью которого пользователь пытается получить доступ к сайту (настольный ПК, телефон или планшет), затем загружает именно ту версию страницы, которая наиболее оптимизирована для него. Для элементов сетки задаются фиксированные в пикселях (px) размеры.

Поэтому основное отличие между этими приёмами — отзывчивый дизайн — один макет для всех устройств, адаптивный дизайн — один макет для каждого вида устройства. Иными словами, сервер берет на себя всю «тяжелую» работу, вместо того, чтобы заставлять сайт оптимизировать самого себя. Среди достоинств адаптивного дизайна можно выделить следующие:

* Изображения загружаются намного быстрее, так как они сжимаются и адаптируются под устройство пользователя
* Загрузка сайта происходит быстрее, так как сервер определяет тип устройства пользователя и загружает соответствующий ему программный код
* Разработчики пользуются свободой творчества, ведь они могут создавать различные версии сайтов и подгонять их под соответствующие типы устройств, чтобы сделать их более удобными для мобильных пользователей.

Привлекательность этого метода омрачается тем, что создать адаптивный сайт не так-то просто. Из-за адаптации дизайна к различным устройствам, время, затрачиваемое на разработку, значительно увеличивается. Более того, если вам потребуется сделать какие-либо доработки на сайте, придется вносить изменения во все его версии.

**Как сервер узнаёт, с какого типа устройства/браузера/ОС/языка вы открываете веб-сайт? (Например, для Adaptive design)**

Когда вы отправляете HTTP-запрос, он содержит в себе заголовки (headers) с различной информацией. Одним из них является User-Agent. Он сообщает: браузер, его версию и язык, движок браузера, версию движка, операционную систему. Данные могут быть написаны как угодно, однако примерный формат таков:

Браузер/Версия (Платформа; Шифрование; Система, Язык[; Что-нибудь еще]) [Дополнения].

Как еще можно определить, если не из хедеров? Определить версию и тип браузера можно при помощи JavaScript.

**Чем отличается авторизация от аутентификации?**

|  |  |
| --- | --- |
| Аутентификация | Авторизация |
| Процедура проверки подлинности субъекта | Процедура присвоения и проверки прав на совершение определенных действий субъектом |
| Зависит от предоставляемой пользователем информации | Не зависит от действий клиента |
| Запускается один раз для текущей сессии | Происходит при попытке совершения любых действий пользователем |

**Как работает авторизация/аутентификация? Как сайт понимает, что ты залогинен?**

Классический вариант – регистрация по логину/почте и паролю. При входе и введении правильных данных, если данные совпадают в таковыми в базе, вы получаете доступ на сайт.

1. Т.к. протокол HTTP не отслеживает состояния, нельзя достоверно знать, что человек, залогинившийся до этого по почте и паролю остается тем же человеком. И тогда изобрели аутентификацию на основе сессий/кук, на основе которых реализовано отслеживание состояний (stateful), то есть аутентификационная запись или сессия хранятся как на сервере, так и на клиенте. Сервер отслеживает открытые сессии в базе данных или в оперативной памяти, в свою очередь на фронтенде создаются cookies, в которых хранится идентификатор сессии. Процедура аутентификации на основе сессий:

* Пользователь вводит в браузере своё имя и пароль, после чего клиентское приложение отправляет на сервер запрос.
* Сервер проверяет пользователя, аутентифицирует его, шлёт приложению уникальный пользовательский токен (сохранив его в памяти или базе данных).
* Клиентское приложение сохраняет токены в куках и отправляет их при каждом последующем запросе.
* Сервер получает каждый запрос, требующий аутентификации, с помощью токена аутентифицирует пользователя и возвращает запрошенные данные клиентскому приложению.
* Когда пользователь выходит, клиентское приложение удаляет его токен, поэтому все последующие запросы от этого клиента становятся неаутентифицированными.

У этого метода есть и недостатки:

* При каждой аутентификации пользователя сервер должен создавать у себя запись. Обычно она хранится в памяти, и при большом количестве пользователей есть вероятность слишком высокой нагрузки на сервер.
* Поскольку сессии хранятся в памяти, масштабировать не так просто. Если вы многократно реплицируете сервер, то на все новые серверы придётся реплицировать и все пользовательские сессии. Это усложняет масштабирование.

2. Аутентификация на основе токенов в последние годы стала очень популярна из-за распространения одностраничных приложений (SPA), веб-API и интернета вещей. Чаще всего в качестве токенов используются Json Web Tokens (JWT). Хотя реализации бывают разные, но токены JWT превратились в стандарт де-факто.

При аутентификации на основе токенов состояния не отслеживаются (stateless). Мы не будем хранить информацию о пользователе на сервере или в сессии и даже не будем хранить JWT, использованные для клиентов.

Процедура аутентификации на основе токенов:

* Пользователь вводит имя и пароль.
* Сервер проверяет их и возвращает токен (JWT), который может содержать метаданные вроде user\_id, разрешений и т. д.
* Токен хранится на клиентской стороне, чаще всего в локальном хранилище, но может лежать и в хранилище сессий или кук.
* Последующие запросы к серверу обычно содержат этот токен в качестве дополнительного заголовка авторизации в виде *Bearer {JWT}*. Ещё токен может пересылаться в теле POST-запроса и даже как параметр запроса.
* Сервер расшифровывает JWT, если токен верный, сервер обрабатывает запрос.
* Когда пользователь выходит из системы, токен на клиентской стороне уничтожается, с сервером взаимодействовать не нужно.

У метода есть ряд преимуществ:

Главное преимущество: поскольку метод никак не оперирует состояниями, серверу не нужно хранить записи с пользовательскими токенами или сессиями. Каждый токен самодостаточен, содержит все необходимые для проверки данные, а также передаёт затребованную пользовательскую информацию. Поэтому токены не усложняют масштабирование.

В куках вы просто храните ID пользовательских сессий, а JWT позволяет хранить метаданные любого типа, если это корректный JSON.

При использовании кук бэкенд должен выполнять поиск по традиционной SQL-базе или NoSQL-альтернативе, и обмен данными наверняка длится дольше, чем расшифровка токена. Кроме того, раз вы можете хранить внутри JWT дополнительные данные вроде пользовательских разрешений, то можете сэкономить и дополнительные обращения поисковые запросы на получение и обработку данных. Допустим, у вас есть API-ресурс /api/orders, который возвращает последние созданные приложением заказы, но просматривать их могут только пользователи категории админов. Если вы используете куки, то, сделав запрос, вы генерируете одно обращение к базе данных для проверки сессии, ещё одно обращение — для получения пользовательских данных и проверки, относится ли пользователь к админам, и третье обращение — для получения данных. А если вы применяете JWT, то можете хранить пользовательскую категорию уже в токене. Когда сервер запросит его и расшифрует, вы можете сделать одно обращение к базе данных, чтобы получить нужные заказы.

У использования кук на мобильных платформах есть много ограничений и особенностей. А токены сильно проще реализовать на iOS и Android. К тому же токены проще реализовать для приложений и сервисов интернета вещей, в которых не предусмотрено хранение кук.

Благодаря всему этому аутентификация на основе токенов сегодня набирает популярность.

Примечание: в целях безопасности в некоторых случаях в дополнение к токену применяется сравнение user agent и подобного. В случае различия вас разлогинит. Так же, например, в банковских системах нельзя одновременно залогиниться в одну учетную запись с нескольких устройств одновременно.

1. Беспарольная аутентификация

Первой реакцией на термин «беспарольная аутентификация» может быть «Как аутентифицировать кого-то без пароля? Разве такое возможно?»

В наши головы внедрено убеждение, что пароли — абсолютный источник защиты наших аккаунтов. Но если изучить вопрос глубже, то выяснится, что беспарольная аутентификация может быть не просто безопасной, но и безопаснее традиционного входа по имени и паролю. Возможно, вы даже слышали мнение, что пароли устарели.

Беспарольная аутентификация — это способ конфигурирования процедуры входа и аутентификации пользователей без ввода паролей. Идея такая:

Вместо ввода почты/имени и пароля пользователи вводят только свою почту. Ваше приложение отправляет на этот адрес одноразовую ссылку, пользователь по ней кликает и автоматически входит на ваш сайт / в приложение. При беспарольной аутентификации приложение считает, что в ваш ящик пришло письмо со ссылкой, если вы написали свой, а не чужой адрес.

Есть похожий метод, при котором вместо одноразовой ссылки по SMS отправляется код или одноразовый пароль. Но тогда придётся объединить ваше приложение с SMS-сервисом вроде twilio (и сервис не бесплатен). Код или одноразовый пароль тоже можно отправлять по почте.

И ещё один, менее (пока) популярный (и доступный только на устройствах Apple) метод беспарольной аутентификации: использовать Touch ID для аутентификации по отпечаткам пальцев. Если вы пользуетесь Slack, то уже могли столкнуться с беспарольной аутентификацией.

Medium предоставляет доступ к своему сайту только по почте. Auth0, или Facebook AccountKit, — это отличный вариант для реализации беспарольной системы для вашего приложения.

Что может пойти не так?

Если кто-то получит доступ к пользовательским почтам, он получит и доступ к приложениям и сайтам. Но это не ваша головная боль — беспокоиться о безопасности почтовых аккаунтов пользователей. Кроме того, если кто-то получит доступ к чужой почте, то сможет перехватить аккаунты в приложениях с беспарольной аутентификацией, воспользовавшись функцией восстановления пароля. Но мы ничего не можем поделать с почтой наших пользователей. Пойдём дальше.

В чём преимущества?

Как часто вы пользуетесь ссылкой «забыли пароль» для сброса пароля, который так и не смогли вспомнить после нескольких неудачных попыток входа на сайт / в приложение? Все мы бываем в такой ситуации. Все пароли не упомнишь, особенно если вы заботитесь о безопасности и для каждого сайта делаете отдельный пароль (соблюдая все эти «должен состоять не менее чем из восьми символов, содержать хотя бы одну цифру, строчную букву и специальный символ»). От всего этого вас избавит беспарольная аутентификация. Знаю, вы думаете сейчас: «Я использую менеджер паролей». Но не забывайте, что подавляющее большинство пользователей не такие техногики, как вы. Это нужно учитывать.

Если вы думаете, что какие-то пользователи предпочтут старомодные логин/пароль, то предоставьте им оба варианта, чтобы они могли выбирать.

Сегодня беспарольная аутентификация быстро набирает популярность.

1. Единая точка входа (Single Sign On, SSO)

Обращали внимание, что, когда логинишься в браузере в каком-нибудь Google-сервисе, например, Gmail, а потом идёшь на Youtube или иной Google-сервис, там не приходится логиниться? Ты автомагически получаешь доступ ко всем сервисам компании. Впечатляет, верно? Ведь хотя Gmail и Youtube — это сервисы Google, но всё же раздельные продукты. Как они аутентифицируют пользователя во всех продуктах после единственного входа?

Этот метод называется единой точкой входа (Single Sign On, SSO).

Реализовать его можно по-разному. Например, использовать центральный сервис для оркестрации единого входа между несколькими клиентами. В случае с Google этот сервис называется Google Accounts. Когда пользователь логинится, Google Accounts создаёт куку, которая сохраняется за пользователем, когда тот ходит по принадлежащим компании сервисам. Как это работает:

* Пользователь входит в один из сервисов Google.
* Пользователь получает сгенерированную в Google Accounts куку.
* Пользователь идёт в другой продукт Google.
* Пользователь снова перенаправляется в Google Accounts.
* Google Accounts видит, что пользователю уже присвоена кука, и перенаправляет пользователя в запрошенный продукт.

Очень простое описание единой точки входа: пользователь входит один раз и получает доступ ко всем системам без необходимости входить в каждую из них. В этой процедуре используется три сущности, доверяющие другу прямо и косвенно. Пользователь вводит пароль (или аутентифицируется иначе) у поставщика идентификационной информации (identity provider, IDP), чтобы получить доступ к поставщику услуги (service provider (SP). Пользователь доверяет IDP, и SP доверяет IDP, так что SP может доверять пользователю.

Выглядит очень просто, но конкретные реализации бывают очень сложными.

1. Аутентификация в соцсетях (Social sign-in) или социальным логином (Social Login). Вы можете аутентифицировать пользователей по их аккаунтам в соцсетях. Тогда пользователям не придётся регистрироваться отдельно в вашем приложении.

Формально социальный логин — это не отдельный метод аутентификации. Это разновидность единой точки входа с упрощением процесса регистрации/входа пользователя в ваше приложение.

Пользователи могут войти в ваше приложение одним кликом, если у них есть аккаунт в одной из соцсетей. Им не нужно помнить логины и пароли. Это сильно улучшает опыт использования вашего приложения. Вам не нужно волноваться о безопасности пользовательских данных и думать о проверке адресов почты — они уже проверены соцсетями. Кроме того, в соцсетях уже есть механизмы восстановления пароля.

Большинство соцсетей в качестве механизма аутентификации используют авторизацию через OAuth2 (некоторые используют OAuth1, например Twitter). Разберёмся, что такое OAuth. Соцсеть — это сервер ресурсов, ваше приложение — клиент, а пытающийся войти в ваше приложение пользователь — владелец ресурса. Ресурсом называется пользовательский профиль / информация для аутентификации. Когда пользователь хочет войти в ваше приложение, оно перенаправляет пользователя в соцсеть для аутентификации (обычно это всплывающее окно с URL’ом соцсети). После успешной аутентификации пользователь должен дать вашему приложению разрешение на доступ к своему профилю из соцсети. Затем соцсеть возвращает пользователя обратно в ваше приложение, но уже с токеном доступа. В следующий раз приложение возьмёт этот токен и запросит у соцсети информацию из пользовательского профиля. Так работает OAuth (ради простоты я опустил технические подробности).

Для реализации такого механизма вам может понадобиться зарегистрировать своё приложение в разных соцсетях. Вам дадут app\_id и другие ключи для конфигурирования подключения к соцсетям. Также есть несколько популярных библиотек/пакетов (вроде Passport, Laravel Socialite и т. д. ), которые помогут упростить процедуру и избавят от излишней возни.

1. Двухфакторная аутентификация (2FA) улучшает безопасность доступа за счёт использования двух методов (также называемых факторами) проверки личности пользователя. Это разновидность многофакторной аутентификации. Наверное, вам не приходило в голову, но в банкоматах вы проходите двухфакторную аутентификацию: на вашей банковской карте должна быть записана правильная информация, и в дополнение к этому вы вводите PIN. Если кто-то украдёт вашу карту, то без кода он не сможет ею воспользоваться. (Не факт! — Примеч. пер.) То есть в системе двухфакторной аутентификации пользователь получает доступ только после того, как предоставит несколько отдельных частей информации.

Другой знакомый пример — двухфакторная аутентификация Mail.Ru, Google, Facebook и т. д. Если включён этот метод входа, то сначала вам нужно ввести логин и пароль, а затем одноразовый пароль (код проверки), отправляемый по SMS. Если ваш обычный пароль был скомпрометирован, аккаунт останется защищённым, потому что на втором шаге входа злоумышленник не сможет ввести нужный код проверки.

Вместо одноразового пароля в качестве второго фактора могут использоваться отпечатки пальцев или снимок сетчатки.

При двухфакторной аутентификации пользователь должен предоставить два из трёх:

То, что вы знаете: пароль или PIN.

То, что у вас есть: физическое устройство (смартфон) или приложение, генерирующее одноразовые пароли.

Часть вас: биологически уникальное свойство вроде ваших отпечатков пальцев, голоса или снимка сетчатки.

Большинство хакеров охотятся за паролями и PIN-кодами. Гораздо труднее получить доступ к генератору токенов или биологическим свойствам, поэтому сегодня двухфакторка обеспечивает высокую безопасность аккаунтов.

И всё же двухфакторка поможет усилить безопасность аутентификации в вашем приложении. Как реализовать? Возможно, стоит не велосипедить, а воспользоваться существующими решениями вроде Auth0 или Duo.

**Почему важно делать подтверждение e-mail при регистрации?**

* Основная причина — это провести double opt-in, т.е. убедиться, что был введен валидный адрес пользователя и этот пользователь действительно дает свое согласие на регистрацию на данном ресурсе и получение дополнительных рассылок/писем. Это позволяет отсеивать «мусорные» регистрации, когда подписывают на что-нибудь посторонних людей (намеренно или нет), уменьшать количество спама (за что очень больно бьют по рукам) и т. д.
* Для защиты страницы. Если кто-нибудь попытается получить доступ к аккаунту пользователя, то на почту пользователя придёт соответствующее уведомление.
* Для быстрого и самостоятельного восстановления доступа (логина и пароля). Многие пользователи испытывают затруднения при восстановлении доступа, если не указали email при регистрации.
* Для отправки электронного чека после оплаты услуг сервиса.
* Для защиты от ботов.
* Если не подтверждать емайл, то в этом поле можно будет написать все что угодно (в рамках проверки). Соответственно один и тот же пользователь будет регистрироваться множество раз, забывая и свой предыдущий пароль, и логин.

**Что такое кэш и зачем его очищать при тестировании?**

Кэш — это временное хранилище для данных (перечень определен создателем сайта) с посещенного сайта. Во-первых, многие элементы на страницах сайта одинаковы: изображения, HTML, CSS, JavaScript и нет смысла каждый раз загружать их заново. Во-вторых, при повторном открытии той же самой страницы действует та же логика – эти элементы уже были загружены, ни к чему грузить их каждый раз по новой.

Сохранение данных веб-страниц на компьютере, вместо их повторной загрузки, помогает экономить время открытия веб-сайтов в браузере и трафик, но с другой стороны снижает срок службы SSD-накопителей, так что вы всегда можете полностью отключить кеширование в своём браузере.

Виды кеширования:

* Кэширование в браузере. Устройство пользователя создает и сохраняет копию различных элементов сайта. Это могут быть: скрипты, текст, изображения и т. д. При открытии страницы кэш браузера помогает загрузить все это в разы быстрее.
* Кэширование на сервере. Все данные хранятся на сервере. Он сохраняет результаты запросов, что помогает избежать повторной обработки одной и той же информации от пользователя.

В тестировании практически во всех случаях правило – очищать кэш после каждого прохода теста (если только это не целенаправленной тестирование самого кэша или требуется наличие кэша по каким-либо причинам). Дело в том, что кэш очевидно искажает показатели performance testing, а также может быть причиной ошибочного дефект-репорта из-за устаревания и/или несогласованности актуальных и сохраненных данных. В некоторых ситуациях без очистки кеша не обойтись даже просто из-за огромного количества кешируемых данных.

**Что такое AJAX в вебе?**

Ajax ( Asynchronous Javascript and XML — «асинхронный JavaScript и XML») — подход к построению интерактивных пользовательских интерфейсов веб-приложений, заключающийся в «фоновом» обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью, и веб-приложения становятся быстрее и удобнее. По-русски иногда произносится транслитом как «аякс». У аббревиатуры AJAX нет устоявшегося аналога на кириллице.

В классической модели веб-приложения:

* Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
* [Браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) формирует и отправляет [запрос](https://ru.wikipedia.org/wiki/HTTP) [серверу](https://ru.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80).
* В ответ сервер генерирует совершенно новую веб-страницу и отправляет её браузеру и т. д. , после чего браузер полностью перезагружает всю страницу.

При использовании AJAX:

* Пользователь заходит на веб-страницу и нажимает на какой-нибудь её элемент.
* [JavaScript](https://ru.wikipedia.org/wiki/JavaScript) определяет, какая информация необходима для обновления страницы.
* [Браузер](https://ru.wikipedia.org/wiki/%D0%91%D1%80%D0%B0%D1%83%D0%B7%D0%B5%D1%80) отправляет соответствующий запрос на [сервер](https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%B5%D1%80_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5)).
* Сервер возвращает только ту часть документа, на которую пришёл запрос.
* [Скрипт](https://ru.wikipedia.org/wiki/%D0%A1%D0%BA%D1%80%D0%B8%D0%BF%D1%82) вносит изменения с учётом полученной информации (без полной перезагрузки страницы).

**Как работает браузер (коротко)?**

Типичный сценарий использования предполагает отправку на некий сервер GET запроса и отображение полученного ответа.

Модуль отображения выполняет синтаксический анализ полученного HTML-документа и переводит теги в узлы DOM(DOM – объектная модель документа (Document Object Model) – служит для представления HTML-документа и интерфейса элементов HTML таким внешним объектам, как код JavaScript.) в дереве содержания. Информация о стилях извлекается как из внешних CSS-файлов, так и из элементов style. Эта информация и инструкции по отображению в HTML-файле используются для создания еще одного дерева – дерева отображения.

Оно содержит прямоугольники с визуальными атрибутами, такими как цвет и размер. Прямоугольники располагаются в том порядке, в каком они должны быть выведены на экран.

После создания дерева отображения начинается компоновка элементов, в ходе которой каждому узлу присваиваются координаты точки на экране, где он должен появиться. Затем выполняется отрисовка, при которой узлы дерева отображения последовательно отрисовываются с помощью исполнительной части пользовательского интерфейса.

**Как работает сотовая связь?**

Почему связь «сотовая»? Если посмотреть сверху на схему сети базовых станций, то их пересекающиеся краями круги покрытия похожи на пчелиные соты.

Сотовая связь потому и называется сотовой, что в основе любой сети — ячейки (соты), каждая сота представляет собой участок территории, который покрывает (обслуживает) базовая станция. Форма и размеры сот зависят от множества факторов, в том числе от мощности излучения базовой станции, стандарта, рабочих частот, направления антенн и т.п. Соты обязательно перекрывают друг друга, это необходимо для того, чтобы мобильное устройство (терминал) не теряло связь при перемещении из одной соты в другую. Особенно это важно для владельца сотового телефона, который разговаривает во время движения.

В условиях городской застройки невозможно разбить карту города на квадратики и поставить базовые станции через равные расстояния, чтобы добиться качественного покрытия. Начинают играть роль этажность застройки, препятствия в виде памятников, возможность установить базовые станции в том или ином месте. Не зря наши города назвали каменными джунглями, планирование в них радиосетей – это та еще задачка. Поэтому все операторы стараются резервировать дополнительные мощности в крупных городах, создавать перекрывающиеся зоны для базовых станций. И этому есть и другая причина.

Для эффективной работы сети одного покрытия мало, базовые станции должны обслуживать одновременно много пользователей. А в городах — очень много одновременно разговаривающих и пользующихся мобильным интернетом. Полосы частот, на которых передаются голос и данные, — ограниченный и крайне ценный ресурс, за их лицензирование операторы во всём мире платят государству большие деньги.

Когда вы набираете номер и начинаете звонить, ну, или вам кто-нибудь звонит, то ваш мобильный телефон по радиоканалу связывается с одной из антенн ближайшей базовой станции. От антенны сигнал по кабелю передается непосредственно в управляющий блок станции. Базовая станция должна выделить вам свободный голосовой канал. Вместе они и образуют базовую станцию [антенны и управляющий блок]. Несколько базовых станций, чьи антенны обслуживают отдельную территорию, например, район города или небольшой населенный пункт, подсоединены к специальному блоку – контроллеру. К одному контроллеру обычно подключается до 15 базовых станций. В свою очередь, контроллеры, которых также может быть несколько, кабелями подключены к «мозговому центру» – коммутатору. Коммутатор обеспечивает выход и вход сигналов на городские телефонные линии, на других операторов сотовой связи, а также операторов междугородней и международной связи.

В небольших сетях используется только один коммутатор, в более крупных, обслуживающих сразу более миллиона абонентов, могут использоваться два, три и более коммутаторов, объединенных между собой опять-таки проводами.

Когда человек передвигается по улице пешком или идет на автомобиле, поезде и т. д. и при этом еще и разговаривает по телефону, важно обеспечить непрерывность связи. Связисты процесс эстафетной передачи обслуживания в мобильных сетях называют термином «handover». Необходимо вовремя переключать телефон абонента из одной базовой станции на другую, от одного контроллера к другому и так далее.

Если бы базовые станции были напрямую подключены к коммутатору, то всеми этими переключениями пришлось бы управлять коммутатору. А ему «бедному» и так есть, чем заняться. Многоуровневая схема сети дает возможность равномерно распределить нагрузку на технические средства. Это снижает вероятность отказа оборудования и, как следствие, потери связи. Итак, достигнув коммутатора, наш звонок переводится далее – на сеть другого оператора мобильной, городской междугородной и международной связи. Конечно же, это происходит по высокоскоростным кабельным каналам связи. Звонок поступает на коммутатор другого оператора. При этом последний «знает», на какой территории [в области действия, какого контроллера] сейчас находится нужный абонент. Коммутатор передает телефонный вызов конкретному контроллеру, в котором содержится информация, в зоне действия какой базовой станции находится адресат звонка. Контроллер посылает сигнал этой единственной базовой станции, а она в свою очередь «опрашивает», то есть вызывает мобильный телефон. Точно также происходят телефонные звонки в разные города России, Европы и мира. Для связи коммутаторов различных операторов связи используются высокоскоростные оптоволоконные каналы связи. Благодаря им сотни тысяч километров телефонный сигнал преодолевает за считанные секунды или даже доли секунд.

**Как работает подключение к Wi-Fi?**

* Начинает процесс подключения клиент, отправляя широковещательное сообщение Обнаружения DHCP (DHCP DISCOVER), в качестве обязательных полей передается номер транзакции - xid, MAC-адрес устройства - chaddr, также в опциях передается последний присвоенный клиенту IP-адрес, однако данная опция может быть проигнорирована сервером.
* Запрос обнаружения рассылается для всех узлов сети, но отвечают на него только DHCP-сервера, формируя сообщение Предложения DHCP (DHCP OFFER), которое содержит предлагаемую сервером сетевую конфигурацию. Если серверов несколько, то предложений клиент получит несколько. Из предложенных конфигураций клиент выбирает одну, как правило полученную первой.
* Так как MAC-адрес отправителя известен, то сервер направляет ответ непосредственно клиенту (unicast), хотя в некоторых случаях может ответить и широковещательным пакетом.
* Приняв предложение, клиент официально запрашивает у сервера данную конфигурацию, для чего отправляет широковещательно Запрос DHCP (DHCP REQUEST), он полностью повторяет по структуре сообщение обнаружения (Discover), только добавляет к нему опцию 54 с адресом сервера, конфигурацию которого клиент принял. Опция 50 содержит предложенный сервером IP-адрес. Несмотря на то, что MAC-адрес DHCP-сервера известен, запрос (Request) рассылается широковещательно, это нужно для того, чтобы остальные DHCP-сервера понимали, что их предложение отвергнуто.
* Получив запрос сервер направляет клиенту в ответПодтверждение DHCP (DHCP ACK), которое отправляется на MAC-адрес клиента (хотя может и широковещательно) и, получив которое, клиент должен настроить свой сетевой адаптер согласно указанному адресу и опциям.
* Получив адрес, клиент может проверить его на предмет использования при помощи широковещательного ARP-запроса (в большинстве реализаций так и происходит) и если будет обнаружено, что выделенный адрес уже используется (скажем, назначен вручную), то клиент посылает широковещательное сообщение ОтказаDHCP (DHCP DECLINE) и начинает процесс получения адреса заново. Сервер, получив сообщение отказа, должен пометить указанный адрес как недоступный и уведомить администратора о возможной проблеме в конфигурации (например, записью в логе).

После этого, все устройства, находящиеся во внутренней сети, будут выходить в Интернет через роутер под одним внешним IP-адресом, но в локальной сети они будут иметь разный IP.

**Базы данных**

* Информация — любые сведения о каком-либо событии, процессе, объекте.
* Данные — это информация, представленная в определенном виде, позволяющем автоматизировать ее сбор, хранение и дальнейшую обработку человеком или информационным средством. Для компьютерных технологий данные — это информация в дискретном, фиксированном виде, удобная для хранения, обработки на ЭВМ, а также для передачи по каналам связи.
* База данных (БД) — именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области, или иначе БД — это совокупность взаимосвязанных данных при такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений в определенной предметной области. БД состоит из множества связанных файлов.
* Система управления базами данных (СУБД) — DBMS — Database management system - совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями. Автоматизированная информационная система (АИС) — это система, реализующая автоматизированный сбор, обработку, манипулирование данными, функционирующая на основе ЭВМ и других технических средств и включающая соответствующее программное обеспечение (ПО) и персонал.
* Банк данных (БнД) является разновидностью ИС. БНД — это система специальным образом организованных данных: баз данных, программных, технических, языковых, организационно-методических средств, предназначенных для обеспечения централизованного накопления и коллективного многоцелевого использования данных.
* Модель – способ структурирования данных, описания взаимосвязей между данными:
* Иерархическая модель. Модель представляет данные в виде иерархии. Модель ориентирована на описание объектов, находящихся между собой в неких отношениях. Например, структура кадров некоторой организации.
* Сетевая модель. Сетевая модель представляет собой развитие иерархической. Модель позволяет описывать более сложные виды взаимоотношений между данными. Однако расширение возможностей достигается за счет большей сложности реализации самой модели и трудности манипулирования данными.
* Реляционная модель. В реляционной модели данные представляются в виде таблиц, состоящих из строк и столбцов. Каждая строка таблицы – информация об одном конкретном объекте, столбцы содержат свойства этого объекта. Взаимоотношения между объектами задаются с помощью связей между столбцами таблиц. Реляционная модель на сегодняшний день наиболее распространена. Она достаточно универсальна и проста в проектировании. Строки таблиц называют записями или кортежами, столбцы – полями или атрибутами. Для того чтобы можно было сослаться на отдельную запись (строку) в некоторой таблице, каждая запись этой таблицы должна содержать уникальный идентификатор. Поле таблицы, значения которого гарантированно уникальны для каждой записи этой таблицы, называют ключевым полем или ключом. Ключ не обязательно должен быть числовым. Иногда уникальным идентификатором может служить не одно поле, а комбинация полей. При этом сочетание значений этих полей должно быть уникальным. Такие поля образуют составной ключ таблицы
* Объектная модель. В этой модели данные представляются в форме объектов. Объект имеет набор свойств, называемых атрибутами, и может включать в себя также процедуры для обработки данных, которые называют методами. Объекты, имеющие одинаковые наборы атрибутов и различающиеся только их значениями, образуют некоторый класс объектов. Например, класс «клиент» может иметь следующие атрибуты: «фамилия», «имя», «отчество», «номер кредитной карты». Для каждого объекта из этого класса определены конкретные значения перечисленных атрибутов. Говорят, что объект является экземпляром класса. На основе существующего класса могут создаваться новые, наследующие свойства исходного. При этом исходный класс именуется родителем нового класса. Производный класс называют потомком исходного. При этом объекты – экземпляры класса-потомка принадлежат также и родительскому классу, поскольку обладают всеми его атрибутами. Пример: на основе класса «клиент» может быть определен класс «постоянный клиент»
* Гибридные модели. В некоторых приложениях предпринимаются попытки смешения различных моделей представления данных. Пример такого смешения – объектно-реляционная модель. В ней использовано некоторое сходство между реляционной и объектной идеологией. Строки таблиц реляционной модели соответствуют объектам объектной модели, столбцы таблиц – атрибутам объектов. Таблицы в целом являются аналогом классов. Отсюда вытекает возможность введения наследования при определении таблиц – таблица-потомок содержит те же столбцы, что и родительская, и, кроме того – дополнительные, определенные при наследовании. По идее создателей, объектно-реляционная модель должна унаследовать от реляционной легкость описания и манипулирования данными, а от объектной – возможность определения более сложных взаимоотношений между объектами.

**Может ли у ПО быть сразу несколько баз данных?**

Может и даже разного типа. Но в простых случаях делать это стоит только когда всё упрется в предел производительности. Начиная с миллиардов записей у одной даже хорошо оптимизированной БД на одной hardware дисковой подсистеме уже могут начаться проблемы с performance, поэтому компания может принять решение разнести одну базу на несколько баз на разных серверах, но вместе с этим появляются вопросы к сетевому аспекту этого решения (задержки и т.п.). Помимо производительности, разделение на несколько БД может быть в угоду безопасности.

**Что такое SQL?**

structured query language — «язык структурированных запросов» — декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.

**Что вы знаете о NoSQL?**

«NoSQL» имеет абсолютно стихийное происхождение и не имеет общепризнанного определения или научного учреждения за спиной. Это название скорее характеризует вектор развития ИТ в сторону от реляционных баз данных. Расшифровывается как Not Only SQL. Базы данных NoSQL специально созданы для определенных моделей данных и обладают гибкими схемами, что позволяет разрабатывать современные приложения. Базы данных NoSQL получили широкое распространение в связи с простотой разработки, функциональностью и производительностью при любых масштабах. Подробнее можно почтитать тут: <https://aws.amazon.com/ru/nosql/>

**Что такое нормальные формы?**

Терминология:

* Атрибут — свойство некоторой сущности. Часто называется полем таблицы.
* Домен атрибута — множество допустимых значений, которые может принимать атрибут.
* Кортеж — конечное множество взаимосвязанных допустимых значений атрибутов, которые вместе описывают некоторую сущность (строка таблицы).
* Отношение — конечное множество кортежей (таблица).
* Схема отношения — конечное множество атрибутов, определяющих некоторую сущность. Иными словами, это структура таблицы, состоящей из конкретного набора полей.
* Проекция — отношение, полученное из заданного путём удаления и (или) перестановки некоторых атрибутов.
* Функциональная зависимость между атрибутами (множествами атрибутов) X и Y означает, что для любого допустимого набора кортежей в данном отношении: если два кортежа совпадают по значению X, то они совпадают по значению Y. Например, если значение атрибута «Название компании» — Canonical Ltd, то значением атрибута «Штаб-квартира» в таком кортеже всегда будет Millbank Tower, London, United Kingdom. Обозначение: {X} -> {Y}.
* Нормальная форма — требование, предъявляемое к структуре таблиц в теории реляционных баз данных для устранения из базы избыточных функциональных зависимостей между атрибутами (полями таблиц).
* Метод нормальных форм (НФ) состоит в сборе информации о объектах решения задачи в рамках одного отношения и последующей декомпозиции этого отношения на несколько взаимосвязанных отношений на основе процедур нормализации отношений. Цель нормализации: исключить избыточное дублирование данных, которое является причиной аномалий, возникших при добавлении, редактировании и удалении кортежей(строк таблицы).
* Аномалией называется такая ситуация в таблице БД, которая приводит к противоречию в БД либо существенно усложняет обработку БД. Причиной является излишнее дублирование данных в таблице, которое вызывается наличием функциональных зависимостей от не ключевых атрибутов.
* Аномалии-модификации проявляются в том, что изменение одних данных может повлечь просмотр всей таблицы и соответствующее изменение некоторых записей таблицы.
* Аномалии-удаления — при удалении какого-либо кортежа из таблицы может пропасть информация, которая не связана на прямую с удаляемой записью.
* Аномалии-добавления возникают, когда информацию в таблицу нельзя поместить, пока она не полная, либо вставка записи требует дополнительного просмотра таблицы.

Нормальные формы:

* Первая нормальная форма: Отношение находится в 1НФ, если все его атрибуты являются простыми, все используемые домены должны содержать только скалярные значения. Не должно быть повторений строк в таблице.
* Вторая нормальная форма: Отношение находится во 2НФ, если оно находится в 1НФ и каждый не ключевой атрибут неприводимо зависит от Первичного Ключа(ПК). Неприводимость означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, от которого можно также вывести данную функциональную зависимость.
* Третья нормальная форма: Отношение находится в 3НФ, когда находится во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа. Проще говоря, второе правило требует выносить все не ключевые поля, содержимое которых может относиться к нескольким записям таблицы в отдельные таблицы.
* Четвертая нормальная форма: Отношение находится в 4НФ, если оно находится в НФБК и все нетривиальные многозначные зависимости фактически являются функциональными зависимостями от ее потенциальных ключей. В отношении R (A, B, C) существует многозначная зависимость R.A -> -> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от С.
* Пятая нормальная форма: Отношения находятся в 5НФ, если оно находится в 4НФ и отсутствуют сложные зависимые соединения между атрибутами. Если «Атрибут\_1» зависит от «Атрибута\_2», а «Атрибут\_2» в свою очередь зависит от «Атрибута\_3», а «Атрибут\_3» зависит от «Атрибута\_1», то все три атрибута обязательно входят в один кортеж. Это очень жесткое требование, которое можно выполнить лишь при дополнительных условиях. На практике трудно найти пример реализации этого требования в чистом виде.
* Шестая нормальная форма: Переменная отношения находится в шестой нормальной форме тогда и только тогда, когда она удовлетворяет всем нетривиальным зависимостям соединения. Из определения следует, что переменная находится в 6НФ тогда и только тогда, когда она неприводима, то есть не может быть подвергнута дальнейшей декомпозиции без потерь. Каждая переменная отношения, которая находится в 6НФ, также находится и в 5НФ. Идея «декомпозиции до конца» выдвигалась до начала исследований в области хронологических данных, но не нашла поддержки. Однако для хронологических баз данных максимально возможная декомпозиция позволяет бороться с избыточностью и упрощает поддержание целостности базы данных.

**Понятие хранимой процедуры?**

Хранимые процедуры представляют собой группы связанных между собой операторов SQL, применение которых делает работу программиста более легкой и гибкой, поскольку выполнить хранимую процедуру часто оказывается гораздо проще, чем последовательность отдельных операторов SQL. Хранимые процедуры представляют собой набор команд, состоящий из одного или нескольких операторов SQL или функций и сохраняемый в базе данных в откомпилированном виде.

**Понятие триггера?**

Три́ггер (англ. trigger) — хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определённом столбце заданной таблицы реляционной базы данных. Триггеры применяются для обеспечения целостности данных и реализации сложной бизнес-логики. Триггер запускается автоматически при попытке изменения данных в таблице, с которой он связан. Все производимые им модификации данных рассматриваются как выполняемые в транзакции, в которой выполнено действие, вызвавшее срабатывание триггера. Соответственно, в случае обнаружения ошибки или нарушения целостности данных может произойти откат этой транзакции.

**Что такое индексы? (Indexes)**

Индекс - объект базы данных, создаваемый с целью повышения производительности поиска данных. Таблицы в базе данных могут иметь большое количество строк, которые хранятся в произвольном порядке, и их поиск по заданному критерию путём последовательного просмотра таблицы строка за строкой может занимать много времени. Индекс формируется из значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы и, таким образом, позволяет искать строки, удовлетворяющие критерию поиска. Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, сбалансированного дерева. Различные типы индексов:

* B-Tree index
* Bitmap index
* Clustered index
* Covering index
* Non-unique index
* Unique index

**Какие шаги выполняет тестер при тестировании хранимых процедур?**

Тестер проверяет стандартный формат хранимых процедур, а также проверяет правильность полей, таких как updates, joins, indexes, deletions как указано в хранимой процедуре.

**Как бы вы узнали для тестирования базы данных, сработал триггер или нет?**

В журнале аудита (audit log) вы можете увидеть срабатывание триггеров.

**Как тестировать загрузку данных при тестировании базы данных?**

* Исходные данные должны быть известны
* Целевые данные должны быть известны
* Совместимость источника и цели должна быть проверена
* В диспетчере SQL Enterprise запустите пакет DTS после открытия соответствующего пакета DTS.
* Вы должны сравнить столбцы цели и источника данных
* Количество строк цели и источника должны быть проверены
* После обновления данных в источнике проверьте, появляются ли изменения в цели или нет.
* Проверьте на NULL и ненужные символы

**Основные команды SQL?**

* Просмотр доступных баз данных

SHOW DATABASES;

* Создание новой базы данных

CREATE DATABASE;

* Выбор базы данных для использования

USE ;

* Импорт SQL-команд из файла .sql

SOURCE ;

* Удаление базы данных

DROP DATABASE ;

* Просмотр таблиц, доступных в базе данных

SHOW TABLES;

* Создание новой таблицы

CREATE TABLE (

<col\_name1> <col\_type1>,

<col\_name2> <col\_type2>,

<col\_name3> <col\_type3>

PRIMARY KEY (<col\_name1>),

FOREIGN KEY (<col\_name2>) REFERENCES <table\_name2>(<col\_name2>)

);

* Добавление данных в таблицу

INSERT INTO (, , , …)

VALUES (, , , …);

При добавлении данных в каждый столбец таблицы не требуется указывать названия столбцов.

INSERT INTO <table\_name>

VALUES (, , , …);

* Обновление данных таблицы

UPDATE

SET <col\_name1> = , <col\_name2> = , ...

WHERE ;

* Удаление всех данных из таблицы

DELETE FROM ;

* Удаление таблицы

DROP TABLE ;

SELECT используется для получения данных из определённой таблицы:

SELECT <col\_name1>, <col\_name2>, …

FROM <table\_name>;

* Следующей командой можно вывести все данные из таблицы:

SELECT \* FROM ;

SELECT DISTINCT

* В столбцах таблицы могут содержаться повторяющиеся данные. Используйте SELECT DISTINCT для получения только неповторяющихся данных.

SELECT DISTINCT , , …

FROM <table\_name>;

WHERE

* Можно использовать ключевое слово WHERE в SELECT для указания условий в запросе:

SELECT , , …

FROM <table\_name>

WHERE ;

* В запросе можно задавать следующие условия:

сравнение текста;

сравнение численных значений;

логические операции AND (и), OR (или) и NOT (отрицание).

Пример

Попробуйте выполнить следующие команды. Обратите внимание на условия, заданные в WHERE:

SELECT \* FROM course WHERE dept\_name=’Comp. Sci.’;

SELECT \* FROM course WHERE credits>3;

SELECT \* FROM course WHERE dept\_name='Comp. Sci.' AND credits>3;

* Оператор GROUP BY часто используется с агрегатными функциями, такими как COUNT, MAX, MIN, SUM и AVG, для группировки выходных значений.

SELECT , , …

FROM <table\_name>

GROUP BY <col\_namex>;

Пример

Выведем количество курсов для каждого факультета:

SELECT COUNT(course\_id), dept\_name

FROM course

GROUP BY dept\_name;

* Ключевое слово HAVING было добавлено в SQL потому, что WHERE не может быть использовано для работы с агрегатными функциями.

SELECT , , ...

FROM <table\_name>

GROUP BY <column\_namex>

HAVING

Пример

Выведем список факультетов, у которых более одного курса:

SELECT COUNT(course\_id), dept\_name

FROM course

GROUP BY dept\_name

HAVING COUNT(course\_id)>1;

* ORDER BY используется для сортировки результатов запроса по убыванию или возрастанию. ORDERBY отсортирует по возрастанию, если не будет указан способ сортировки ASC или DESC.

SELECT , , …

FROM <table\_name>

ORDER BY <col\_name1>, <col\_name2>, … ASC|DESC;

Пример

Выведем список курсов по возрастанию и убыванию количества кредитов:

SELECT \* FROM course ORDER BY credits;

SELECT \* FROM course ORDER BY credits DESC;

* BETWEEN используется для выбора значений данных из определённого промежутка. Могут быть использованы числовые и текстовые значения, а также даты.

SELECT , , …

FROM <table\_name>

WHERE <col\_namex> BETWEEN AND ;

Пример

Выведем список инструкторов, чья зарплата больше 50 000, но меньше 100 000:

SELECT \* FROM instructor

WHERE salary BETWEEN 50000 AND 100000;

* Оператор LIKE используется в WHERE, чтобы задать шаблон поиска похожего значения.

Есть два свободных оператора, которые используются в LIKE:

% (ни одного, один или несколько символов);

\_ (один символ).

SELECT <col\_name1>, <col\_name2>, …

FROM <table\_name>

WHERE <col\_namex> LIKE ;

Пример

Выведем список курсов, в имени которых содержится «to», и список курсов, название которых начинается с «CS-»:

SELECT \* FROM course WHERE title LIKE ‘%to%’;

SELECT \* FROM course WHERE course\_id LIKE 'CS-\_\_\_';

* С помощью IN можно указать несколько значений для оператора WHERE:

SELECT , , …

FROM <table\_name>

WHERE <col\_namen> IN (, , …);

Пример

Выведем список студентов с направлений Comp. Sci., Physics и Elec. Eng.:

SELECT \* FROM student

WHERE dept\_name IN (‘Comp. Sci.’, ‘Physics’, ‘Elec. Eng.’);

* JOIN используется для связи двух или более таблиц с помощью общих атрибутов внутри них.
* View — это виртуальная таблица SQL, созданная в результате выполнения выражения. Она содержит строки и столбцы и очень похожа на обычную SQL-таблицу. View всегда показывает самую свежую информацию из базы данных.

Создание

CREATE VIEW <view\_name> AS

SELECT <col\_name1>, <col\_name2>, …

FROM <table\_name>

WHERE ;

Удаление

DROP VIEW <view\_name>;

Пример

Создадим view, состоящую из курсов с 3 кредитами:

* 24. Агрегатные функции — эти функции используются для получения совокупного результата, относящегося к рассматриваемым данным. Ниже приведены общеупотребительные агрегированные функции:

COUNT (col\_name) — возвращает количество строк;

SUM (col\_name) — возвращает сумму значений в данном столбце;

AVG (col\_name) — возвращает среднее значение данного столбца;

MIN (col\_name) — возвращает наименьшее значение данного столбца;

MAX (col\_name) — возвращает наибольшее значение данного столбца.

* Вложенные подзапросы — это SQL-запросы, которые включают выражения SELECT, FROM и WHERE, вложенные в другой запрос.

Пример

Найдём курсы, которые преподавались осенью 2009 и весной 2010 годов:

SELECT DISTINCT course\_id

FROM section

WHERE semester = ‘Fall’ AND year= 2009 AND course\_id IN (

SELECT course\_id

FROM section

WHERE semester = ‘Spring’ AND year= 2010

);

**Подробнее о джойнах? (Join)**

Как и было сказано выше, различные виды JOIN помогают объединить некие данные из нескольких таблиц каким-либо образом.

Так чем отличается INNER JOIN от LEFT JOIN? Чаще всего ответ примерно такой: «inner join — это как бы пересечение множеств, т.е. остается только то, что есть в обеих таблицах, а left join — это когда левая таблица остается без изменений, а от правой добавляется пересечение множеств. Для всех остальных строк добавляется null». Еще, бывает, рисуют пересекающиеся круги.

Это понимание и подобные ответы – по сути не верны, т.к. все джойны – декартово произведение (cross join) с фильтрами (предикатом и, возможно, UNION). Советую углубиться в тему, например, тут или там:

<https://habr.com/ru/post/448072/>

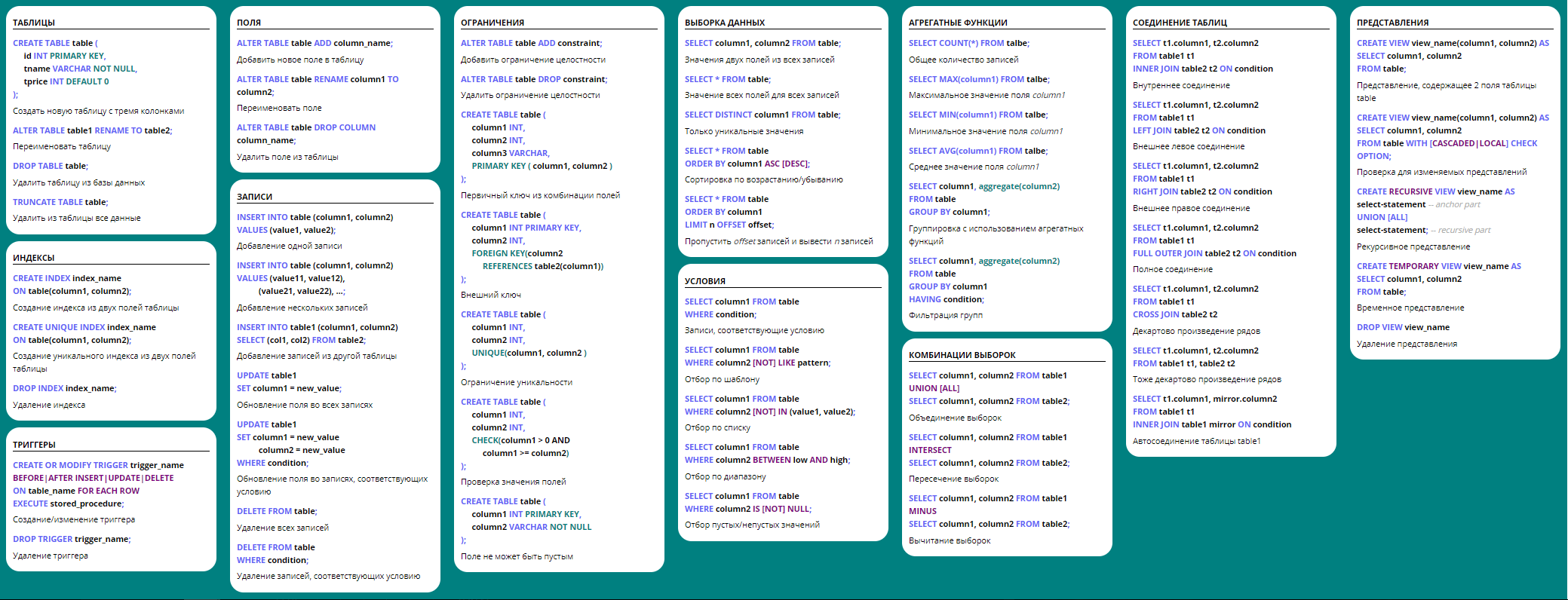
<https://akorsa.ru/2016/09/kak-na-samom-dele-rabotayut-operatsii-join-v-sql/>

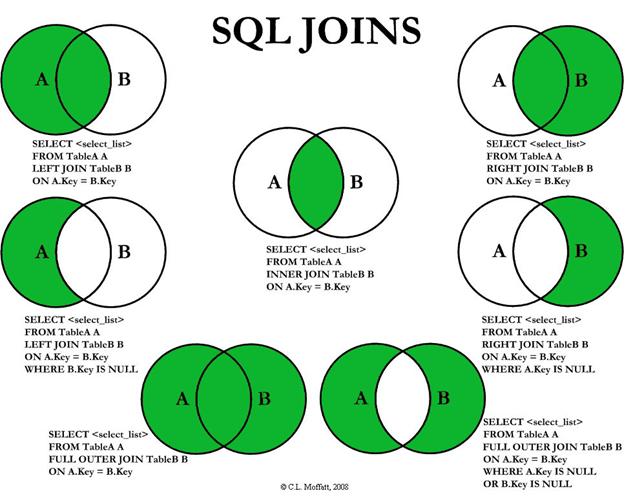
Так же стоит обратить внимание на порядок таблиц при различных джойнах.

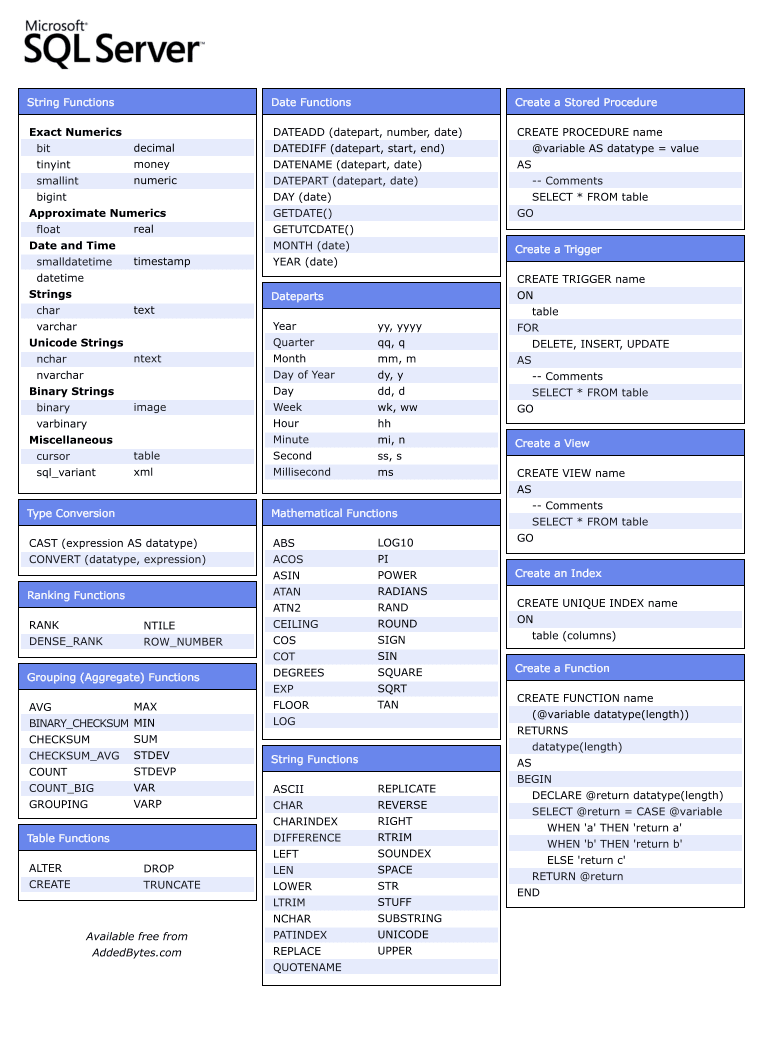
**Типы данных в SQL?**

* Exact Numeric SQL Data Types:
  + bigint = Range from -2^63 (-9,223,372,036,854,775,808) to 2^63-1 (9,223,372,036,854,775,807) int = Range from -2^31 (-2,147,483,648) to 2^31-1 (2,147,483,647) smallint = Range from -2^15 (-32,768) to 2^15-1 (32,767) tinyint = Range from 0 to 255 bit = 0 and 1 decimal = Range from –10^38 +1 to 10^38 -1 numeric = Range from -10^38 +1 to 10^38 -1 money = Range from -922,337,203,685,477.5808 to +922,337,203,685,477.5807 small money = Range from -214,748.3648 to +214,748.3647
* Approximate Numeric SQL Data Types:
  + float = Range from -1.79E + 308 to 1.79E + 308 real = Range from -3.40E + 38 to 3.40E + 38
* Date and Time SQL Data Types:
  + datetime = From Jan 1, 1753 to Dec 31, 9999 smalldatetime = From Jan 1, 1900 to Jun 6, 2079 date = To store a date like March 27, 1986 time = To store a time of day like 12:00 A.M.
* Character Strings SQL Data Types:
  + char = Maximum length of 8,000 characters varchar = Maximum of 8,000 characters varchar(max) = Maximum length of 231 characters text = Maximum length of 2,147,483,647 characters.
* Unicode Character Strings SQL Data Types:
  + nchar = Maximum length of 4,000 characters nvarchar = Maximum length of 4,000 characters nvarchar(max) = Maximum length of 231 characters ntext = Maximum length of 1,073,741,823 characters
* Binary SQL Data Types:
  + binary = Maximum length of 8,000 bytes varbinary � = Maximum length of 8,000 bytes varbinary(max) = Maximum length of 231 bytes image = Maximum length of 2,147,483,647 bytes

**Шпаргалки SQL**

[](https://camo.githubusercontent.com/17f0910a750ae5a573f923cb1409b6aa211cd8e00982b9e421908613cb6d5a7b/68747470733a2f2f686162726173746f726167652e6f72672f776562742f72782f6f692f6e722f72786f696e726231786a6873766677776669676d793363706f786f2e706e67)

[](https://camo.githubusercontent.com/4895c02bc2a05d1beb583cb74236d59b8204ef198a2489f7c5b8184146d1ab2a/68747470733a2f2f686162726173746f726167652e6f72672f776562742f76342f766b2f74782f7634766b7478662d7a656574775f7a6b706f736c6c797363646b772e706e67)

[](https://camo.githubusercontent.com/7ecb0396ee863098f2b0779a27e3fc424dd902fdf70a0c87df1bfce047ae8fe8/68747470733a2f2f686162726173746f726167652e6f72672f776562742f6a6b2f31632f64692f6a6b316364697a6b6c316473397769797971786e67756d6b7133302e706e67)

**ПРАКТИКА**

**Дана форма для регистрации. Протестируйте.**

Вопрос номер один практически на всех собеседованиях на младшую позицию. Он хорош еще и тем, что в зависимости от уровня кандидата будет раскрыт в разной степени. Всегда в первую очередь уточняйте хотя бы какие-то минимальные требования, даже если вначале озвучивают, что требования не формализованы.

* Начальный уровень представляет из себя простые позитивные и негативные кейсы (в основном на валидацию):
  + Обязательные поля отмечены \*
  + Обязательные поля заполнены/нет
  + Галочки на соглашениях проставлены/нет
  + Поле password и подтверждение имеет соответствующий тип (в полях формы прописан корректный атрибут TYPE, сообщающий браузеру тип элементов формы.)
  + Проверяется, что пароли одинаковы
  + Имя пользователя валидируется как минимум на длину и спец. символы, остальное по ТЗ
  + Адрес почты валидируется в соответствии со стандартом (наличие символа @, несколько символов @, длины частей до и после @, допустимые символы до и после, наличие пробелов перед адресом и после, корректная доменная часть и т.п.)
  + Поля с ожидаемым числовым вводом и текстовым соответственно проверить позитивными и негативными кейсами по типам данных
* Следующий уровень:
  + Всё из предыдущего
  + Кроссбраузерность
  + Понятность формы. Присутствует описание полей или плейсхолдеры
  + Сенситив данные не должны передаваться в URL
  + Проверяем, как форма отображается до сабмита и после
  + Поведение, если нажать сабмит несколько раз подряд
  + Если формы очищаются после сабмита, проверить регистрацию существующего пользователя
  + Проверка глобализации – номер телефона, дата, почтовый индекс, валюта, вертикальное или RTL письмо и т.п. (опционально)
  + Проверка простых инъекций
  + Правильная работа многошаговых форм (Навигация рядом с формой показывает текущий этап и количество оставшихся шагов.)
  + Для полей, предполагающих загрузку файлов, прописан атрибут accept, определяющий тип загружаемых документов
  + Текстовое многострочное поле при вводе объемного сообщения изменяет высоту либо в правой части появляется скроллбар для просмотра всего содержимого
  + Для авторизованного пользователя в поля формы автоматически подставляются все известные о посетителе данные.
  + Форма сохраняется в веб-формах (админ-панели) или SQL-таблицах.
  + Прописан реальный e-mail лица, отвечающего за обработку заявок (если предполагается ОС)
  + Опционально. Пользователь получает уведомление на свой e-mail об успешно полученной заявке и последующих действиях, которые от него требуются.
  + Прописан атрибут autocomplete для полей, поддерживающих это значение
* Extra:
  + Проверяем, отправились ли данные после сабмита
  + Проверяем, добавились ли соответствующие записи в бд
  + Проверка загрузки формы и сабмита при медленном/нестабильном интернет-соединении
  + Корректность cookies/токена и т.п. после сабмита

**Определение серьезности и приоритета**

Очень частый вопрос на собеседованиях. Либо вам дают конкретные примеры дефектов, для которых вы должны определить серьезность и приоритет, либо вас самих просят придумать варианты дефектов в каких-либо ситуациях. Например, дверь в магазине. Какой дефект может быть с низкой серьезностью, но высоким приоритетом? У каждой компании найдутся свои варианты вопросов, так что потренируйтесь заранее.

**Определение граничных значений и классов эквивалентности**

Следующий по популярности вопрос. Зная азы этих техник тест-дизайна, ответить довольно просто, но всё равно будьте внимательнее. Потренироватся можно на просторах интернета.

**Логические задачи**

Могут быть буквально на логику:

«Саша смотрит на Ольгу, а Ольга смотрит на Андрея. У Саши есть дети, у Андрея нет. Смотрит ли человек, у которого есть дети, на человека, у которого детей нет? Варианты ответа: «Да», «Нет», «Нельзя определить». Объясните свою точку зрения.»

На рассуждение и перебор вариантов из области computer science:

* Мессенджер. Один пользователь отправляет другому сообщение – не доходит. В чем может быть причина?
* Два абсолютно идентичных компьютера (аппаратная и программная конфигурация), файлы скачиваются с разной скоростью. Почему?
* Два абсолютно идентичных компьютера (аппаратная и программная конфигурация), на одном баг воспроизводится, на другом нет. Почему?
* Два разных мобильных устройства с одинаковой версией приложения. Бэк и связь стабильны. На одну приходят нотификации, на другую нет. В чем может быть причина?

Или на «логику»:

«Есть две изолированные друг от друга комнаты. В одной находятся 3 лампочки, в другой — три выключателя. Вы стоите в комнате с выключателями и можете перейти в комнату с лампочками лишь один раз. Необходимо определить, какая лампочка включается каким выключателем.»

К первому типу можно подготовиться, изучив самые азы мат. логики и порешав несколько примеров. Многие относятся к этому несерьезно и проваливают этот тип заданий, между тем такие задачки щелкают на олимпиадах пятиклашки.

Второй тип задач показывает эрудированность в области computer science, здесь помогут только базовые общие курсы.

Про подготовку к третьему типу задач, если опустить дискуссии об их бесполезности, можно сказать только то, что проще их просто прочитать и запомнить решение. Подробнее изучить тему можно в популярной книге «Как сдвинуть гору Фудзи».

**Еще примеры**

* Дан веб-сайт, на котором есть каталог и реализована регистрация. На каких уровнях и что будете тестировать, конкретно по пунктам?
* Дана баг трекинговая система. Протестируйте воркфлоу (жизненный цикл бага).
* Outlook — протестировать форму отправки письма (только этот функционал).
* Дано мобильное приложение: случайное подбрасывание игрального кубика. Как будете тестировать (кейсы)?
* Могут попросить накидать кейсов и в другом направлении. Например, придумайте кейсы для метода замены строки.
* Возможно не актуально в СНГ, но в англоязычных ресурсах встречаются задачи на decision/statement/branch coverage
* Сайт для тренировки тестирования <http://testingchallenges.thetestingmap.org/index.php>

**Набор небольших задач по SQL**

Потренироваться можно тут:  
<https://www.learnqa.ru/sql_test>  
<https://www.sql-ex.ru/index.php?Lang>  
<https://www.db-fiddle.com/>  
Дана база:

CREATE TABLE Departments

([Id] int, [Name] varchar(100))

;

CREATE TABLE Employees

([Id] int, [Name] varchar(100), [Salary] int, [ManagerId] int, [DepartamentId] int)

;

INSERT INTO Departments

([Id], [Name])

VALUES

(1, 'Sales'),

(2, 'Development')

;

INSERT INTO Employees

([Id], [Name], [Salary], [ManagerId], [DepartamentId])

VALUES

(1, 'Ivanov', 100000, null, 1),

(2, 'Petrov', 120000, 1, 1),

(3, 'Sidorov', 130000, 2, 1),

(4, 'Korotkov', 120000, 2, 1),

(5, 'Filev', 90000, 1, 1),

(6, 'Smirnov', 125000, null, 2),

(7, 'Godov', 125000, null, null)

/*1. Фамилии и зарплаты всех сотрудников*/ select Name, Salary from Employees

/*2. Фамилии и зарплаты всех сотрудников, отсортированные по зарплате по возрастанию*/ select Name, Salary from Employees order by Salary asc

/*3. Фамилии и зарплаты всех сотрудников, отсортированные по зарплате по убыванию*/ select Name, Salary from Employees order by Salary desc

/*4. Фамилии и зарплаты всех сотрудников, у которых зарплата больше 100000*/ select Name, Salary from Employees where Salary > 100000

/*5. Фамилии и зарплаты всех сотрудников, у которых зарплата равна 100000*/ select Name, Salary from Employees where Salary = 100000

/*6. Фамилии и зарплаты всех сотрудников, у которых зарплата больше либо равна 100000*/ select Name, Salary from Employees where Salary >= 100000

/*7. Фамилии и зарплаты всех сотрудников, у которых фамилия Ivanov*/ select Name, Salary from Employees where Name = ‘Ivanov’

/*8. Ид департамента и максимальная зарплата в этом департаменте*/ select DepartamentId, max(Salary) from Employees group by DepartamentId

/*9. Имя сотрудника и название его отдела*/ select emp.Name, dep.Name from Employees as emp join Departments as dep on dep.Id = emp.DepartamentId

/*10. Имя сотрудника и имя его начальника*/ select emp2.Name, emp1.Name from Employees as emp1 join Employees as emp2 on emp1.Id = emp2.ManagerId

/*11. Добавить сотрудника с именем Semenov, зарплатой 70000, менеджером Ivanov и отделом Sales*/ insert into Employees ([Id], [Name], [Salary], [ManagerId], [DepartamentId]) values (8, ‘Semenov’, 70000, 1, 1)

/*12. Изменить зарплату сотрудника Godov на 80000*/ update Employees set Salary = 80000 where Id = 7

/*13. Удалить сотрудника Godov*/ delete from Employees where Id = 7

**Тестирование чашки для кофе**

Не уверен, что это еще популярно спрашивать, но на всякий случай пара примеров с просторов.

Сначала – позитивное тестирование.

Функции чашки:

* вмещать напитки,
* переносить напитки,
* возможность пить из нее,
* возможность греть напитки в микроволновке.

Проверим сначала «вмещать»: Поставили на поверхность – стоит, не падает — все ок. Холодной воды налили – вода внутри — все ок. Кипятку налили — не треснула, не течет – все ок. \*сперва хотела лить только горячую (раз горячую выдержит, то холодную – само собой), но потом решила, что это будет заодно и стрессовое тестирование (испытание на перепад температур).

Теперь – «переносить»: Есть ручка, за нее можно взяться и поднять/перенести даже полную и горячую чашку (пустую и холодную носить не станем, если предусмотрен более тяжелый тест).

Теперь – «пить из нее»: Наклонять удается, отхлебывать – тоже. Все ок.

«Возможность греть в микроволновке»: Если в инструкции к чашке не указана максимальная допустимая температура при подогреве в печи, наливаем воду, ставим чашку в печь и включаем на максимум

По идее, нужно ставить таймер на время, достаточное для нагрева напитка до 100 градусов. Потому что если он выкипит, а чашка перегреется, это уже не будет позитивным тестированием.

Если позитивное тестирование удачно, проведем негативное (ошибочные или нестандартные, но возможные действия с предметом): Подвергнем ее

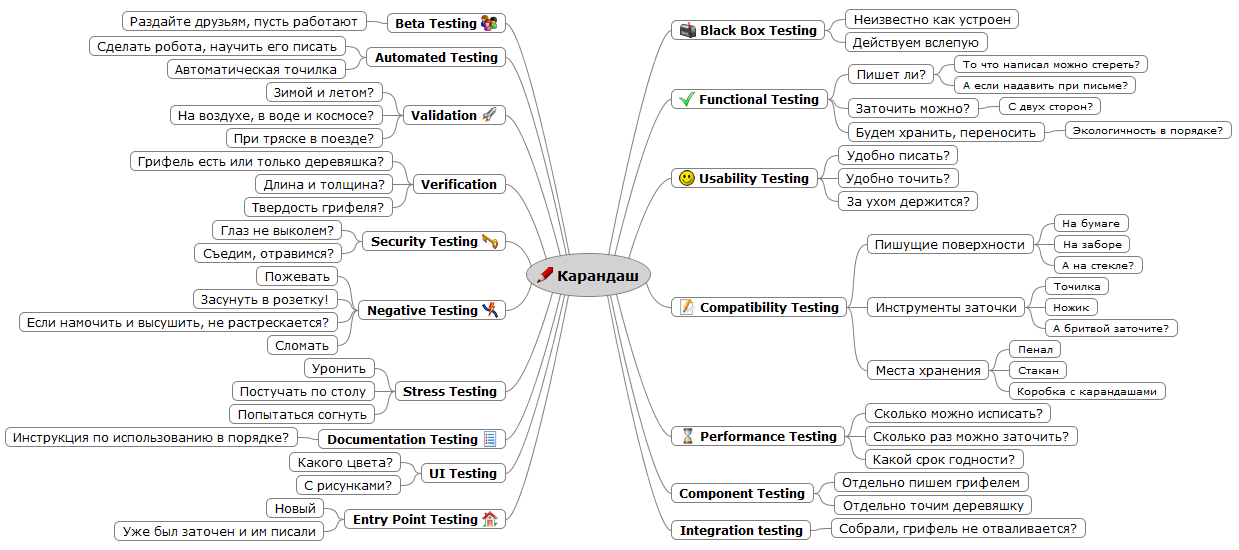
* механическому (об пол, ап стену),
* химическому (кротом, адриланом, уксусной кислотой),
* физическому воздействию (перегреем в микроволновке, поставим на раскаленную горелку).

Еще — «тестирование удобства пользователя»: удобно ли ставить, не горячо ли носить, приятно ли браться за ручку и т. д.

Еще – «тестирование безопасности»: Вот на работе у меня чашка – маленькая, и край отбитый. Поэтому ее никто не трогает, когда меня нет. А эту наверняка все таскали бы – большая, удобная, красивая… Мне не жалко, но тест безопасности она бы не прошла.

Еще – «тестирование взаимодействия»: встает ли чашка на блюдца от других сервизов.

Тестирование карандаша

[](https://camo.githubusercontent.com/5cd666d00b555b4a934f5ec06c35bce9f2cd43421d11845008183acb79c2f2b2/68747470733a2f2f686162726173746f726167652e6f72672f776562742f70792f31792f346b2f70793179346b5f33656a6d707977747a3739696639656a676864692e706e67)

**HR: Как вы будете решать конфликты между членами вашей команды?**

А также еще уйма подобных каверзных вопросов, в ответах на которые вам нужно проявить чудеса маневрирования, менеджерских качеств, софт-скилов и т.п. В общем-то не всегда можно угадать, что от вас ожидают услышать, но в целом это вопросы на оценку вашей адекватности, умения работать в коллективе и прочих софт-скилов.

**HR: Что делать, если разработчик утверждает, что найденный дефект таковым не является?**

Указывать на требования, аппелировать к здравому смыслу, подключать аналитика, чтобы объяснил. Если это поведение не описано в доке, то это баг, либо недоработка. Но недоработка в терминах джиры все равно баг

Проверить ТЗ. Если есть расхождение с ожидаемым результатом – привязываем ссылку на ТЗ.

Если формально это не зафиксировано, но вы чувствуете, что на это стоит обратить внимание – идётё к писателю/аналитику/менеджеру, объясняете и в случае согласия это попадает в ТЗ.

Если формально не зафиксировано и менеджер с вами не согласен – дефект закрывается.

**Вот тебе комп и работающий сайт. Сделай мне 401-ю ошибку.**

Задача на умение пользоваться инструментами, позволяющими подменять трафик. По обстоятельствам.

**С чего начать абсолютному новичку?**

**Путь**

Я бы посоветовал выбрать по отзывам хороший базовый курс и пройти его. Кто-то начинает не с курсов, а какой-то классики типа Савина или Куликова. После поверхностного ознакомления нужно понять свой дальнейший путь – roadmap. Такие карты есть для всех ключевых профессий и тестирование — не исключение. Вот пара ссылок на карты, которые мне понравились, но можете на просторах найти и другие.  
<https://github.com/anas-qa/Quality-Assurance-Road-Map> <https://www.mindmeister.com/ru/1324282825/junior-qa?fullscreen=1> <https://www.mindmeister.com/ru/1558647509?t=973hdS2AKb>  
Некоторые компании подробно расписывают на своих порталах ожидания от каждой стадии развития сотрудника, тоже не проблема найти, по этой же теме много видео на Youtube. Еще один ориентир – просто открыть и почитать вакансии в своём городе, что в среднем у вас требуется от новичка.

Тестирование – самая широкая область в IT. Т.к. теория хоть и не сильно сложная, но ее настолько много, что невозможно изучить всё, нужно пытаться как можно быстрее найти применение своим навыкам. Начать стоит с классики типа тестирования форм, тренировочных сайтов с дефектами специально для тестировщиков и т.п. Не стоит забывать и о софт скилах и базовой грамотности. Бич многих современных тестировщиков даже высокого уровня – безграмотность. Вы можете джуном быть более привлекательны для команды только за счет умения хорошо и правильно излагать свои мысли. Вас также выделит пытливый ум – грош цена тому тестировщику, что напишет куцый саммари, прикрепит непонятный скриншот и назначит дефект. Умение и желание раскапывать причину дефекта (насколько хватит навыков) сильно выделит вас на фоне многих коллег.

По мере роста компетенций как можно раньше стоит начать проходить собеседования и пытаться устроиться на любую стажировку, вообще любой вариант, где вы сможете применять знания и указать этот опыт в резюме, т.к. без опыта сейчас найти работу очень трудно. Если нет никаких оффлайн вариантов, как было у меня, можете регистрироваться на краудтестинговых платформах, искать в тг-каналах возможности потестировать какие-то проекты за бесплатно (иногда там ищут волонтеров за опыт) и т.п.

Когда вы устроитесь на свою первую работу, спустя некоторые время сможете начать готовиться к дальнейшему развитию и выбору направления, ведь никто не заставляет всю жизнь быть ручным тестировщиком. Вы можете сосредоточиться на mobile\web\desktop платформе, профессионально развиваться в менеджеры или автоматизацию, готовиться к узкой специализации — безопасности или performance и т. д., а также сфокусироваться на подготовке по перспективным направлениям:

* ML&AI в QA
* QAOps
* Тестирование IoT
* Тестирование больших данных

Помимо прочего, специалисту, планирующему развиваться профессионально, желательно как можно раньше начать сначала посещать релевантные митапы и конференции, а со временем и выступать в роли докладчика. Также не лишними будут различные сертификации (хотя бы тот же ISTQB разных уровней). Больше о подготовке к ISTQB: habr.com/ru/post/334358 istqb-training.ru

**CV**

Кратко о базовых рекомендациях.

Резюме стажера или джуниора – ровно 1 страница (имеется в виду вариант в файле, а не на площадках). Помимо PDF желательно иметь вордовскую копию на google-диске. Язык резюме – русский, если нацелены на компании из РФ с клиентами из РФ. В остальных случаях – английский. Лучший вариант оформления шапки:

Просто нормальное фото, ФИО, на какую позицию претендуете, опционально локация и дата рождения, актуальные контакты.

После чего идет раздел с опытом работы (любые практические навыки), где максимально кратко и ёмко описывается чем конкретно вы занимались. Это самая важная часть резюме.

Следом – образование и затем ключевые навыки. Не забудьте упомянуть знание иностранных языков. Сориентироваться поможет, например, бесплатный тест EFSET с сертификатом.

Раздел «О себе» можно включить, если есть что важного или интересного написать, опять же, коротко и если есть чем выделиться. В иных случаях это можно включить в сопроводительное письмо при отклике (которое, думаю все уже знают, нужно писать в 100% случаев и подгонять по ситуации).

Никогда не используйте банальные ключевые навыки «ответственный, целеустремленный, …». Только конкретика. Помните, что HR часто ищут по ключевым словам, а вы не должны раздувать ваше резюме всяким мусором. Технологии, инструменты – хороший выбор. Но будьте готовы, что вас по ним детально будут спрашивать в первую очередь.

**Собеседование**

Главное, что нужно сказать о собеседованиях – на них надо ходить. Регулярно. Это отдельный навык, который быстро утрачивается если не практиковать. Не стоит бояться и относиться к ним как к экзамену, вы и сами это поймёте на практике. В знаниях это скорее это как калибровка, нахождения ориентира где вы сейчас находитесь и в какую сторону корректировать курс. Хорошее сравнение собеседований с холодными звонками в продажах. Прошли – проанализировали – подкачались – повторяете, пока не будет достигнут приемлемый результат. Некоторые советуют даже после устройства на работу периодически ходить на собеседования, чтобы держать этот навык в тонусе и ориентироваться в своей ценности на рынке. Аналогия с холодными звонками здесь еще и в том, что слать резюме стоит не только откликаясь на вакансии. Есть мнение, что это уже тупик, когда компания уже выкинула вакансию на работные сайты. Шлите на почты компаний, HR-ов и т.д. Слышал, что активные джуны рассылали по несколько сотен писем в неделю. Стоит ли говорить, что они уже давно устроились?

Если вы ищете первую работу, начать ходить на собеседования нужно как можно раньше еще и потому, что никто не задумывается, как долго в реальности может занять процесс поиска работы. Сначала нужно будет выбить себе возможность пройти интервью. Цифры примерно такие: 10 откликов на подходящие вакансии или 100 рассылкой = 1 собес. 10 не заваленных собесов = 1 оффер. Принятие решения компанией может занимать пару недель. В некоторых компаниях этапов может быть штук до 6. Средне-оптимистичный сценарий предполагает от 1-2 месяцев с нуля до начала работы, но это в случае наличия вокруг выбора и хорошей подготовки у кандидата (а на этот счет тоже многие заблуждаются). В не идеальном случае процесс займёт от полугода.

Основное, что хочется сказать про само собеседование:

Узнай о компании, в которую идешь проходить собеседование.

Не скромничай и не занижай свои навыки. Ты, возможно, 50-й кандидат на этой неделе и еще 100 будет после тебя. Если будешь мяться – про тебя забудут, как только ты закроешь за собой дверь. Здоровая гипербола и немного красок еще никому не мешали. Но никогда не переходи в ложь.

Потренируйтесь в самопрезентации, записывая себя на видео и пересматривая. Обычно первое, что вас спросят – «расскажите о себе». Основная задача HR – проверить адекватность кандидата, в некоторых случаях еще соответствие определенному заказу из целевой команды (от возраста до хобби, всё что угодно). Помните, что собеседуют в компанию в первую очередь человека, а уже потом специалиста. Кто-то сравнивает собеседование со свиданием, где за час вы должны понять, подходите ли вы друг другу. Вопросы на этом этапе в основном стандартные, ответы на них лучше расписать заранее, но не заучивать. Заученный ответ обычно выглядит нелепо, а вот сформулировать свои мысли заранее и понять себя бывает полезно.

Хороший базовый рассказ о себе определит дальнейший ход собеседования, HR будет копать вглубь от заинтересовавших фактов. Кстати, не на все вопросы вы будете знать ответ и это нормально. Одна из задач рекрутера проверить, как вы себя ведете и как отвечаете, когда не знаете ответ.

В конце (хотя бывает и в начале) спрашивает собеседуемый! Помните, что это обоюдный процесс? Они выбирают себе кандидата, вы выбираете себе компанию, обе стороны заинтересованы закрыть торги прямо здесь и сейчас. Немного информации для борьбы со стеснением: для компании найм сотрудника очень затратная затея. Это оплата работы HR (обычно только на собеседования уходит от месяца), рекламы, оборудование рабочего места, введение в процессы, время ментора (посмотрите зарплаты сеньоров, которые будут няньчить новичков первое время). В крупных компаниях даже есть бонусы сотрудникам за удачную рекомендацию знакомого в размере тысяч 100 и это в контексте затрат на весь процесс найма сотрудника просто копейки. По поводу спросить – вот безжалостный копипаст (есть мнение, что если всё это начнет спрашивать начинающий специалист, то на это покрутят у виска. Просто умело и ненавязчиво вплетайте самое важное для вас в беседу и всё будет ок):

* Белая ли заработная плата? Предусмотрены ли премии?
* Есть ли KPI, что в них входит?
* Работа осуществляется по трудовому договору?
* Какой график работы, как происходят отработки?
* Как часто бывают переработки и оплачиваются ли они?
* Приветствуется ли инициатива и если нет, насколько она наказуема?
* Есть ли командировки, какие направления и как часто?
* Что входит в социальный пакет и когда он предоставляется?
* Если работа далеко от дома — как организован транспорт?
* Если работа удалённая или частично удалённая — какая техника предоставляется?
* Как организовано рабочее место, что в него входит? Опенспейс или кабинет?
* Есть ли наставничество или менторство в первое время работы в компании?
* Как осуществляется контроль за сотрудниками: есть ли тайм-трекеры, камеры и т. д.?

Еще: как дела с опозданиями, тренажерка, душ, столовка, литература, ноут, сертификация, курсы? Есть ли фаерволл, схема карьерного роста, роль на проекте?

Если на собеседовании несколько иерархически подчинённых человек (HR и директор, начальник отдела и директор или топ-менеджер), обратите внимание на модель их общения, на то, слаженно ли они работают, если ли контакт или же только благоговейное молчание. Команду видно издалека.

* Как отнеслись к вашему резюме: оно одно из многих и вы здесь «на потоке» или оно лежит одно, и вы в фокусе.
* Как распределено время на собеседование и часто ли отвлекаются его участники, вовремя ли начато общение или вам пришлось ждать больше 15 минут.
* Как вам представили руководителя — с регалиями или нет, по имени-отчеству или имени, формально или неформально.
* Как отнеслись к ходу решения вами заданий — как к экзамену или как к деловому обсуждению задачи. Это говорит о вашем уровне в глазах собеседующего.

Собеседование на английском языке практическое такое же\*, просто из-за языкового барьера могут возникать трудности. Практикуйтесь! И помните, что ваш собеседник не всегда нейтив и ему может быть так же трудно, как и вам.

\*- при собеседовании в другую страну следует учитывать культурные особенности (да и законодательство), потому что некоторые ценности и взгляды могут совершенно не очевидным образом быть разными. Здесь нужно целенаправленно читать статьи о найме или релокации в интересующую страну, там упоминаются эти нюансы и особенности.

**Ошибки в работе у начинающих тестировщиков**

* Во всем видят дефекты. Как избежать:
  + Внимательно анализировать требования
  + Владеть информацией о том, как должен работать продукт
  + Если сомневаешься, что это дефект – спроси БА или ответственного
  + Несколько раз перепроверь прежде чем регистрировать дефект
* Пытаются сразу всё сломать. Как избежать:
  + Начинать тестирование только с положительных тестов
  + Акцентировать внимание на том, что в приоритете для заказчика
  + Не проверять редкие сценарии в первую очередь
* Боятся задавать вопросы. Как избежать:
  + Начать понимать, что коммуникация это важная и неотъемлемая часть работы
* Не интересуются, кто и за что отвечает, как устроены процессы. Как избежать:
  + Узнать у ПМ об областях ответственности каждого члена команды
  + Узнать у ПМ о всех процессах на проекте
* Паникуют при малейшей трудности. Как избежать:
  + Без паники, п. 3 и 4 помогут разобраться
* Пытаются применить сразу всё, что изучили. Как избежать:
  + Помнить, что у каждого вида тестирования своя цель
  + Бюджет всегда ограничен, расставлять приоритеты
* Задают один и тот же вопрос несколько раз

**Полезное**

**Youtube-каналы**

Mikhail Portnov: <https://www.youtube.com/channel/UCDbzkNMBNZJBKP4C-9qGw4Q>

Тестирование: <https://www.youtube.com/channel/UC9VeXtf7fcCJUfmZ_cyweXA>

Heisenbug: <https://www.youtube.com/channel/UCX6fjZa167tSy_4ryTLcOBw>

QA START UP: <https://www.youtube.com/channel/UCAlCZby7zJHzyhOmS8issDQ>

Компания DINS: <https://www.youtube.com/channel/UCmJYB3hvbF3AlVh9HFeXX3A>

Fest Group: <https://www.youtube.com/channel/UClTnsvgTiW2YcfP1tcI2oKA>

QAGuild: <https://www.youtube.com/channel/UCHtyBZ2XbtsRmNiAxh48RGg>

IT conf (SQA days): <https://www.youtube.com/user/vldcorp/videos>

**Telegram**

Каналы это: знакомство с коммьюнити, живое общение; богатая история сообщений, где поиском по истории сообщений можно найти всё что угодно; в шапке каждого канала закреплено сообщение со своим набором полезностей. А еще, заметных в обсуждениях толковых новичков иногда хантят в стажеры ☺

<https://t.me/qajuniors>

<https://t.me/qa_ru>

<https://t.me/qa_courses>

<https://t.me/qa_resumes>

<https://t.me/qa_interviews>

<https://t.me/qa_fin>

<https://t.me/booksqa>

<https://t.me/heisenbugconf>

**Web**

<https://www.tutorialspoint.com/software_testing_dictionary/index.htm>

<https://software-testing.ru/> (+форум)

[https://www.guru99.com](https://www.guru99.com/)

<https://www.softwaretestingmaterial.com/>

<https://www.ministryoftesting.com/>

<https://www.learnqa.ru/>

<https://devhints.io/>

<https://slothman.dev/>

<https://httpbin.org/>

<https://reqres.in/>

**Книги**

Ничего нового не перечислю, углубиться можно в соответствующих tg-каналах.

* Роман Савин «Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах» (совсем поверхностная книга, «для чайников»)
* Святослав Куликов «Тестирование программного обеспечения. Базовый курс.»
* Арбон Джейсон, Каролло Джефф, Уиттакер Джеймс «Как тестируют в Google»
* Борис Бейзер «Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем»
* Рекс Блэк «Ключевые процессы тестирования»
* Гленфорд Майерс, Том Баджетт, Кори Сандлер «Искусство тестирования программ.»
* [Microsoft Corporation](https://www.amazon.com/s/ref=dp_byline_sr_book_1?ie=UTF8&field-author=Microsoft+Corporation&text=Microsoft+Corporation&sort=relevancerank&search-alias=books) — Performance testing Guidance for Web Applications