

Наталія Матвеева

ШПАРГАЛКА начинаючого ТЕСТИРОВЩИКА



Версія 28766790.008
2022

СОДЕРЖАНИЕ

Предисловие	4
Введение	
Что такое тестирование	5
Баг.....	7
UX/UI-дизайн	8
Жизненный цикл ПО (SDLC)	9
Методологии и модели разработки ПО	10
<i>Водопадная модель</i>	10
<i>V-образная модель</i>	11
<i>Итерационная инкрементальная модель</i>	12
<i>Спиральная модель</i>	13
<i>Гибкая методология разработки</i>	14
Scrum и Canban	15
User Story	16
Жизненный цикл тестирования (STLC)	17
Виды тестирования	
Классификация тестирования	18
<i>По запуску кода на исполнение</i>	19
Тестирование документации и требований	20
<i>По доступу к коду</i>	21
<i>По степени автоматизации</i>	22
<i>По уровню детализации приложения</i>	24
<i>По степени важности тестируемых функций</i>	25
<i>По позитивности сценариев</i>	26
<i>В зависимости от целей тестирования</i>	27
Функциональное тестирование	27
Нефункциональное тестирование	27
Связанное с изменениями	28
Локализация vs интернационализация	29
Юзабилити тестирование	30
Тестирование доступности	30
Тестовая документация	
Виды тестовой документации	31
Чек-лист	31
Тест-кейс и его жизненный цикл	32
Оценка трудозатрат.	34

Тест-план	34
Тест-репорт. Метрики. ПМИ	35
Отчет о дефектах (Баг-репорт)	36

Техники тестирования (Тест-дизайн)

Тест-дизайн	37
Классы эквивалентности и граничные значения	37
Техника состояний и переходов	38
Тестирование с помощью таблиц решений	38
Попарное тестирование	39
Предположение об ошибке	39
Исследовательское тестирование	40
Ad-hoc тестирование	40
Monkey тестирование	40
Мобильное тестирование	41
<i>Чек-лист тестирования</i>	44

Основы баз данных. SQL запросы

Основы баз данных	46
SQL запросы	48

API тестирование

Основы API	49
HTTP Protocol	49
Клиент-серверная архитектура	51
Тестирование API	52

HTML, CSS для тестировщика

Введение в HTML и CSS	53
Язык HTML	53
Язык CSS	55

Дополнения

Инструменты тестировщика	56
Примеры вопросов на собеседовании	58

ПРЕДИСЛОВИЕ

Меня зовут Наталия, и я написала эту Шпаргалку. Создавала её, в первую очередь, для себя любимой.

Цель была **структурировать** полученные знания, а также, иметь возможность быстро **найти ответы** на вопросы.

Шпаргалка является компиляцией информации из множества источников, как с рунета, так и с англоязычных сайтов.

Девиз, под эгидой которого я её создавала – **«Не истина в последней инстанции, но триггер вам в помощь»**.

Картинки из сети старалась не комуниздить, а рисовать сама, т.к. качественных, собственно, их и нет в инете. Найденные тексты сокращала и приближала к понятному, но цензурному, русскому – тоже сама. Отдельное спасибо камраду – Инне Когиновой за оперативную правку!

Проект некоммерческий, но буду рада донату троим своим детям на мороженое:

 **ВТБ** 5368 2902 0086 3191

Если вы не собираетесь учить меня, как мне жить, то найти меня можно здесь:



<https://vk.com/zona97>

– Вы выбрали IT из-за того, что там зарплаты высокие, или у вас искренний интерес?

– У меня искренний интерес к высокой зарплате.



ТЕСТИРОВАНИЕ

Тестирование (Software Testing) – проверка соответствия реальных и ожидаемых результатов поведения программы, проводимая на конечном наборе тестов, выбранном определённым образом.

Цели тестирования:

- Выявление дефектов до того, как их обнаружат пользователи.
- Предоставление актуальной информации о состоянии продукта на данный момент.
- Проверка на соответствие ПО всем заявленным требованиям.

Принципы тестирования:

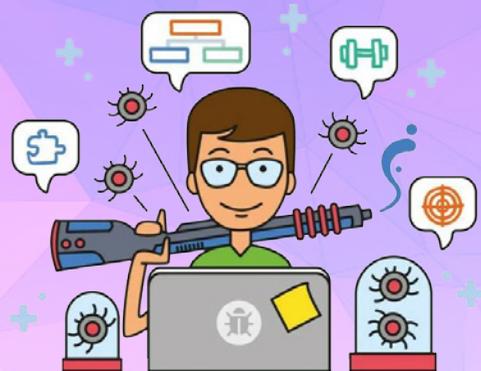
- **Тестирование демонстрирует наличие дефектов.** Тестирование только снижает вероятность наличия дефектов, которые находятся в ПО, но не гарантирует их отсутствия.
- **Исчерпывающее тестирование невозможно.** Полное тестирование с использованием всех входных комбинаций данных, результатов и предусловий физически невыполнимо.
- **Раннее тестирование.** Следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО, чтобы найти дефекты как можно раньше.
- **Скопление дефектов.** Большая часть дефектов находится в ограниченном количестве модулей.
- **Эффект (парадокс) пестицида.** Если повторять те же тестовые сценарии снова и снова, в какой-то момент этот набор тестов перестанет выявлять новые дефекты.

Суть его в том, что если вы долго проводите одни и те же проверки, скорее всего новых багов вы не найдете. Именно поэтому периодически нужно «встряхивать» тестовую базу, ревьюить её новыми сотрудниками, проводить исследовательское тестирование.

- **Тестирование зависит от контекста.** Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем новостной портал.
- **Заблуждение об отсутствии ошибок.** Отсутствие найденных дефектов при тестировании не всегда означает готовность продукта к релизу. Система должна быть удобна пользователю в использовании и удовлетворять его ожиданиям и потребностям.

ТЕСТИРОВЩИК

Тестировщик – это специалист, который проверяет качество программного обеспечения и уровень его соответствия заранее определённым требованиям.

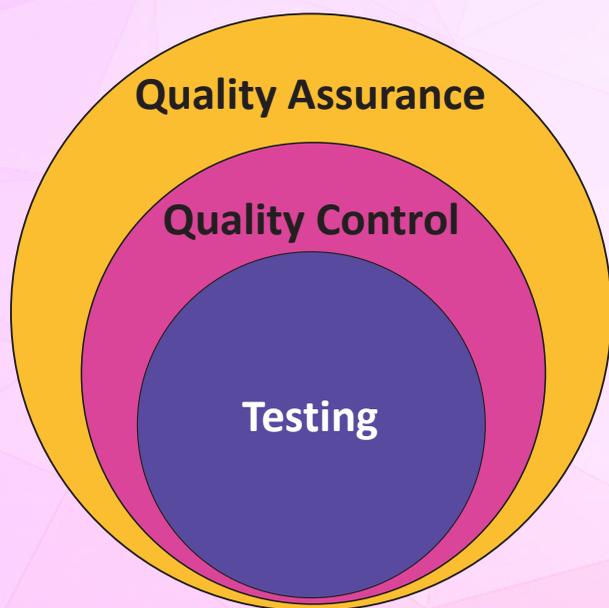


QA И ТЕСТИРОВАНИЕ

Quality Assurance (QA) – QA обеспечивает правильность процесса тестирования, подходит к контролю качества глобально, следит за нормализацией процессов.

Quality Control (QC) – QC предполагает контроль соблюдения требований.

Testing – Тестировщик обеспечивает сбор данных, которые вносятся в документы, созданные в ходе работы QC.



Атрибуты качества ПО:

- Функциональные возможности (Functionality).
- Надёжность (Reliability).
- Практичность (Usability).
- Эффективность (Efficiencies).
- Сопровождаемость (Maintainability).
- Мобильность (Portability).

В реальной жизни IT-индустрии встречаются только два названия профессии QA-инженер и Тестировщик ПО.

Причём очень часто эти понятия взаимозаменяются и путаются. **Пример:** «Ищу Тестировщика ПО (QA-инженера)». По факту многие работодатели ищут тестировщика ПО (если ориентироваться по описанию обязанностей), но в названии вакансии может быть QA-инженер.



Тестировщик ищет баги и сообщает о них. Но главная задача тестировщика – предоставить информацию о том, как работает приложение.

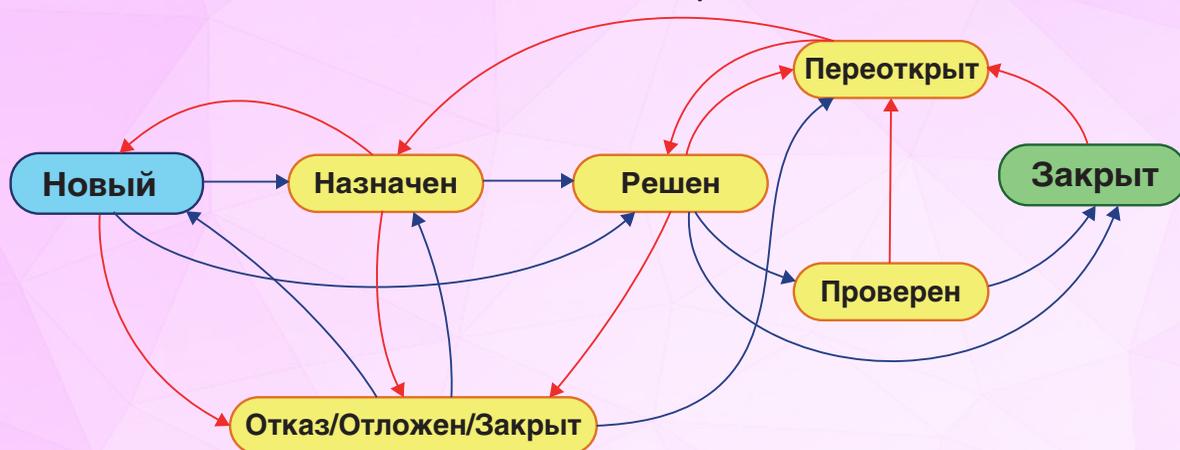
БАГ

БАГ (bug, дефект) – отклонение фактического результата от ожидаемого. *Найденные баги оформляются в баг-репорты (стр. 36).*

Ожидаемый результат (Expected result) – описание того, как именно должна работать система в соответствии с документацией.

Фактический результат (Actual result) – это тот результат, который получает тестировщик во время тестирования. То есть то, как система работает на самом деле.

ЖИЗНЕННЫЙ ЦИКЛ БАГА



- **Новый** (New) – Впервые найденный баг, занесённый в систему.
- **Отказ** (Rejected) – Баг отклонён. Причины: некачественное описание или такой дефект уже существует, невозможность воспроизвести баг.
- **Закрыт** (Closed). – Баг перестал быть актуальным.
- **Назначен** (Assigned) – Дефект просмотрен и открыт, то есть признан для исправления.
- **Решен** (Fixed) – Дефект исправили, и он в этом состоянии требует перепроверки тестировщиком.
- **Переоткрыт** (Re-opened) – Если дефект не исправлен или исправлен не полностью.



ОШИБКА – действие, приводящее к некорректным результатам – ошибки в коде при разработке.

ДЕФЕКТ (баг) – скрытый недостаток в ПО, возникший из-за ошибки в написании кода. Дефект, обнаруженный тестировщиком называется багом.

СБОЙ (отказ) – упущенный при тестировании дефект, который обнаружил пользователь.



Баг vs Фича

Если **БАГ** – это недосмотр, то **ФИЧА** – специально предусмотренная возможность.

Популярная фраза «*Не баг, а фича*» заключается в том, что **баг** – **случайно допущенная ошибка**, мешающая нормально пользоваться программой, а **фича** – **специально сделанная функция**, необычная для других программ такого типа.

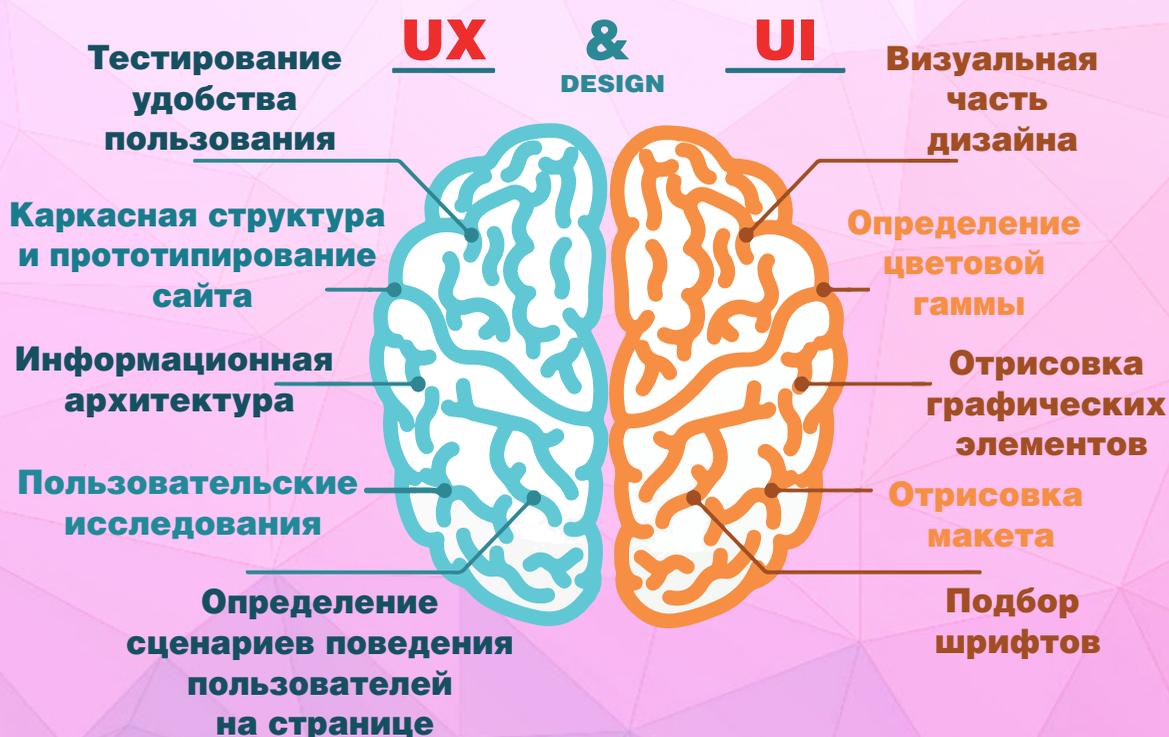
UX/UI ДИЗАЙН

UX (User Experience – опыт пользователя). То, какой опыт/впечатление получает пользователь от работы с интерфейсом. Удастся ли ему достичь цели, и насколько просто/сложно это сделать.

UI (User Interface – пользовательский интерфейс). То, как выглядит интерфейс и то, какие физические характеристики приобретает. Определяет, какого цвета будет ваше «изделие», удобно ли будет человеку попадать пальцем в кнопки, читабельным ли будет текст и тому подобное.

UX/UI дизайн – проектирование любых пользовательских интерфейсов, в которых удобство использования так же важно, как и внешний вид.

Прямая обязанность UX/UI дизайнера – это «продать» товар или услугу через интерфейс. Именно на основе работы UX/UI дизайнера пользователь принимает решение: «Быть или не быть?» Нравится или не нравится. Купить или не купить.



UX дизайнер планирует то, как вы будете взаимодействовать с интерфейсом, и какие шаги вам нужно предпринять, чтобы сделать что-то. А UI дизайнер придумывает, как каждый из этих шагов будет выглядеть.

ЖИЗНЕННЫЙ ЦИКЛ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ (SDLC)

Жизненный цикл ПО (Software development lifecycle) – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.



ЖИЗНЕННЫЙ ЦИКЛ РАЗРАБОТКИ:

- **Анализ требований** отвечает на вопрос «Какие проблемы требуют решений?».
- **Планирование** отвечает на вопрос «Что мы хотим сделать?».
- **Проектирование и дизайн** отвечает на вопрос «Как мы добьемся наших целей?».
- **Разработка ПО** регулирует процесс создания продукта.
- **Тестирование** регулирует обеспечение качественной работы продукта.
- **Развертывание** регулирует использование финального продукта.



SDLC определяет все стандартные фазы, которые участвуют в процессе разработки программного обеспечения, тогда как процесс **STLC** определяет различные действия для улучшения качества продукта.

STLC – это жизненный цикл тестирования, стр. 17.

МОДЕЛИ И МЕТОДОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методология разработки ПО – набор методов и критериев оценки, которые используются для постановки задачи, планирования, контроля и для достижения поставленной цели.

Сам процесс разработки описывается **моделью**, которая определяет последовательность наиболее общих этапов и получаемых результатов.

Модель жизненного цикла ПО – структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач на протяжении жизненного цикла.

Модель выбирают исходя из:

- *Направления проекта*
- *Масштаба проекта*
- *Бюджета*
- *Сроков реализации конечного продукта*

ВОДОПАДНАЯ МОДЕЛЬ

Waterfall Model – подразумевает последовательное прохождение стадий, каждая из которых должна завершиться полностью до начала следующей.

Особенности:

- *Разработка проходит быстро.*
- *Стоимость и сроки заранее определены.*
- *Хороший результат только в проектах с четко и заранее определенными требованиями.*
- *Нет возможности сделать шаг назад.*



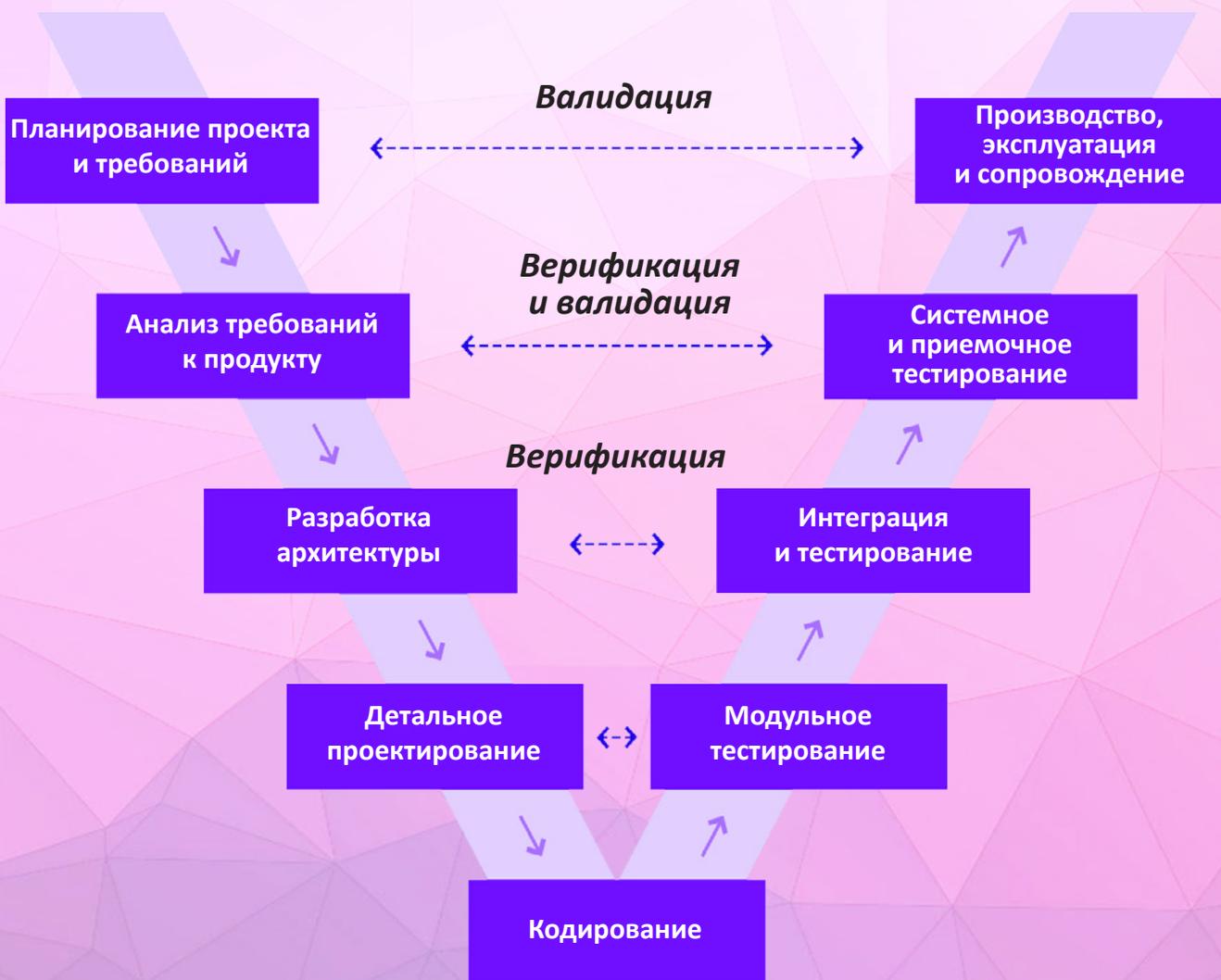
V-ОБРАЗНАЯ МОДЕЛЬ

V-Model – Унаследовала структуру «шаг за шагом» от водопадной модели. Применима к системам, которым особенно важно бесперебойное функционирование. Например, прикладные программы в клиниках для наблюдения за пациентами, интегрированное ПО для механизмов управления аварийными подушками безопасности в транспортных средствах и т.д.

V-образную модель ещё называют разработкой через тестирование.

Особенности:

- Направлена на тщательную проверку и тестирование продукта с ранних стадий проектирования.
- Стадия тестирования проводится одновременно с соответствующей стадией разработки.
- Количество ошибок в архитектуре ПО сводится к минимуму.
- Если при разработке архитектуры была допущена ошибка, то вернуться и исправить её будет стоить дорого, как и в «водопаде».



ИТЕРАЦИОННАЯ ИНКРЕМЕНТАЛЬНАЯ МОДЕЛЬ

Iterative and incremental development

ИТЕРАТИВНОСТЬ – повторение операций в целях переработки результатов предыдущего этапа. **ИНКРЕМЕНТИРОВАНИЕ** – приращение результатов предыдущего этапа.

Проект при этом подходе в каждой фазе развития проходит повторяющийся цикл (итерация):

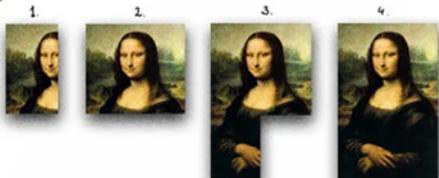
Планирование – Реализация – Проверка – Корректировка.

Преимущества:

- Гибкость в принятии новых требований или изменений .
- Возможность адаптации процесса на основе уроков, извлеченных из предыдущих итераций.
- Более короткие сроки вывода продукта на рынок.

Недостатки:

- Стоимость продукта неизвестна
- Могут возникнуть проблемы с архитектурой системы, поскольку требования для всего жизненного цикла программы не собираются.



ИНКРЕМЕНТАЛЬНАЯ МОДЕЛЬ



ИТЕРАТИВНАЯ МОДЕЛЬ

СПИРАЛЬНАЯ МОДЕЛЬ

Spiral Model – это модель процесса разработки ПО с учетом рисков. Это комбинация модели водопада и итеративной модели. Spiral Model помогает внедрить элементы разработки программного обеспечения из нескольких моделей процессов для программного проекта на основе уникальных шаблонов рисков, обеспечивая эффективный процесс разработки.

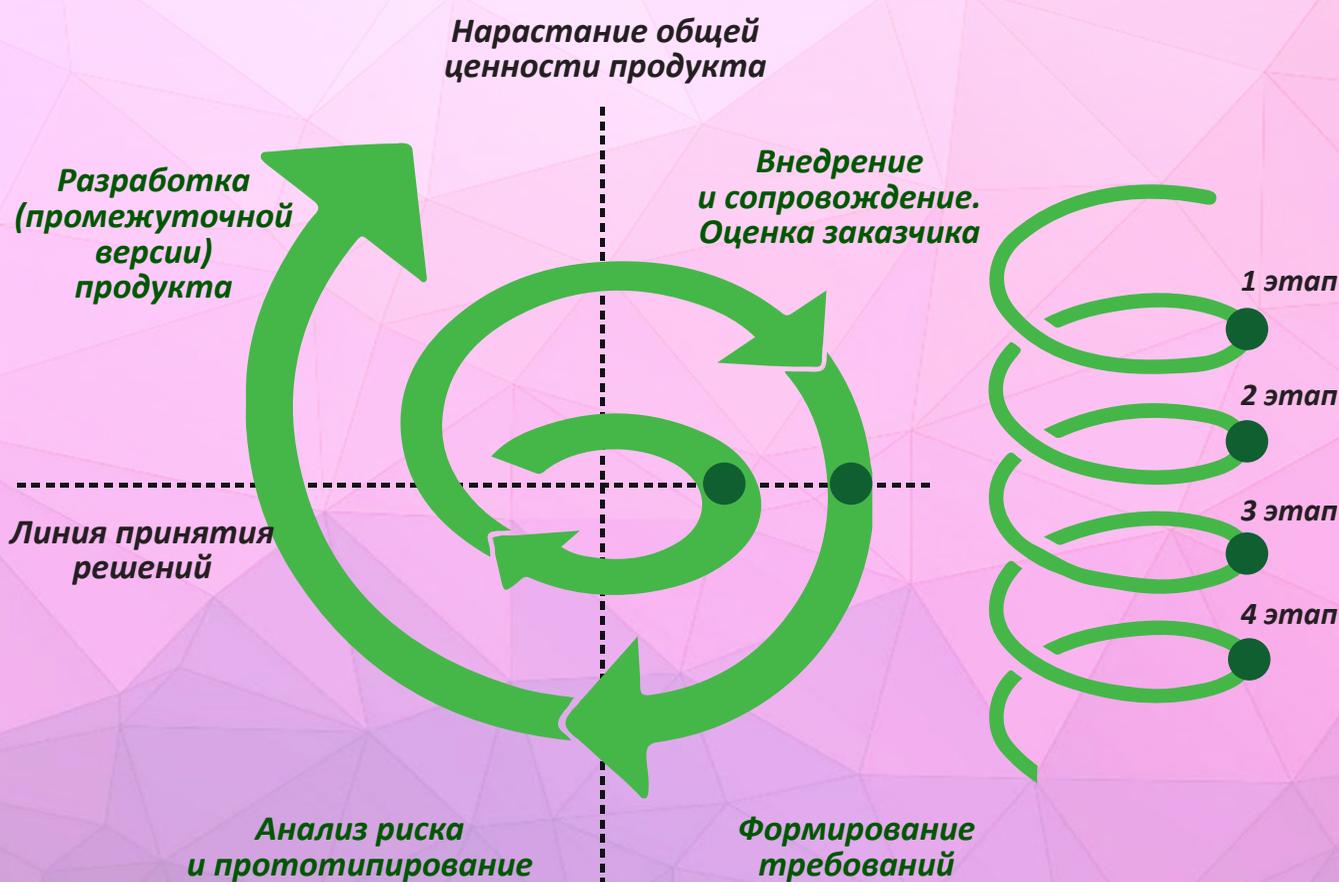
Каждая фаза (итерация) спиральной модели в разработке программного обеспечения начинается с определения цели проектирования и заканчивается тем, что клиент просматривает прогресс.

Преимущества:

- Анализ рисков и управление рисками на каждом этапе.
- Подходит для больших проектов.
- Возможность изменения в требованиях на поздних этапах.
- Заказчик может наблюдать за развитием продукта на ранней стадии разработки.

Недостатки:

- Спиральная модель намного сложнее других моделей SDLC.
- Не подходит для небольших проектов, так как она дорогая.
- Успешное завершение проекта зависит от анализа рисков.
- Количество этапов неизвестно в начале проекта



ГИБКАЯ МЕТОДОЛОГИЯ РАЗРАБОТКИ

Методология Agile – это набор практик, целью которых является оперативная реакция на изменения в ходе рабочего процесса. Такие подходы помогают командам быстро реагировать на обратную связь от клиентов и заказчиков, тем самым постоянно улучшая производимый продукт.

Ценности Agile

- *Люди важнее процессов и стандартных инструментов;*
- *Создание продукта важнее, чем подбор и согласование документации;*
- *Услышать позицию клиента и сотрудничать с ним выше контракта;*
- *Вносить коррективы важнее, чем изначально разработанная стратегия.*

Суть AGILE МАНИФЕСТА:

- Вся работа над проектом разделяется на короткие циклы (итерации) и ведется поэтапно;
- В конце каждой итерации заказчик получает готовый минимально работающий продукт или его часть, которую уже можно использовать;
- В течение всего рабочего процесса команда сотрудничает с заказчиком;
- Любые изменения в проекте приветствуются и быстро интегрируются в работу.



Проще говоря, **Agile** — это своеобразная философия, но как ее придерживаться, зависит от конкретного случая. Есть 2 основных фреймворка, которые основываются на базовых принципах Agile: **Scrum** и **Kanban**.



Да, вы разработчик, и да, вы гибкий, но это не делает вас гибким разработчиком.

SCRUM

Подход предполагает взаимодействие на каждом этапе разработки ПО, не только команды разработчиков (*Delivery Team*), но и самого заказчика (*Product owner*), а также *Scrum-мастера*.

Суть данного фреймворка:

- Работа делится на спринты длительностью 1-3 недели.
- Перед началом спринта команда сама формирует список задач на итерацию.
- Во время каждого спринта создается продукт или услуга, которые можно продемонстрировать клиенту.
- После выполнения спринта проводится ретроспектива. Это митинг, цель которого получить фидбэк от каждого участника команды, выявить текущие успехи и проблемы, то есть оценка и анализ проделанной работы.
- Каждый последующий этап будет наращивать функционал проекта, пока все функции не будут реализованы.

Scrum подходит для объемных проектов.

КАНБАН

Канбан – это способ правильного выстраивания процесса с целью максимально эффективного использования возможности каждого сотрудника. Подход позволяет оптимизировать работу команды через разделение объемных этапов на отдельные операции.

Система Канбан основана на принципах

- Визуализация. Основа Канбан – визуальная доска, на которой представлены этапы выполнения текущей задачи: «запланировано» / «выполняется» / «сделано».
- Разделение задач. Четкое понимание целей упрощает процесс.
- Фокусировка на работе. Невыполненные задачи требуют особого внимания. Если процесс затягивается, нужно подключать дополнительных сотрудников и перераспределять ресурсы.



Scrum vs Kanban

- **Основная разница между Scrum и Канбан – в длине итераций. В Scrum итерации – в среднем 2 недели, в Kanban задачи программисту можно подкидывать хоть каждый день.**
- **Если в Scrum цель команды – закончить спринт, то в Kanban – задачу.**

USER STORY В МЕТОДОЛОГИИ AGILE

User Story – это приём записи требований, который помогает понять нужду клиента и после обсуждения **выбрать**, описать и утвердить **то решение**, которое удовлетворит эту нужду.

Для чего применяется User Story?

- Для описания элементов **бэклога**.
- Для лучшего понимания пользователей.
- Для описания требований к продукту на понятном для всех языке: пользователей, разработчиков другие заинтересованных лиц.
- Для вовлечения в процесс разработки пользователей и заинтересованных лиц.
- Для построения **User Story Mapping**.

Главное в пользовательской истории – это ценность, которую пользователь получит от функции.

Текст самой **user story** должен объяснять роль/действия юзера в системе, его потребность и профит, который юзер получит после того как история случится

История должна быть написана по такому образцу:

Как <роль пользователя>, **я** <что-то хочу получить>, <с такой-то целью> (Как пациент, я хочу созваниваться с врачами по видеосвязи, чтобы иметь возможность обсудить вопросы здоровья).

User Story можно оценить по критериям «INVEST»:

Independent – независимая от других историй, то есть истории могут быть реализованы в любом порядке.

Negotiable – обсуждаемая, отражает суть, а не детали; не содержит конкретных шагов реализации.

Valuable – ценная для клиентов, бизнеса и стейкхолдеров.

Estimable – оцениваемая по сложности и трудозатратам.

Small – компактная, может быть сделана за одну итерацию.

Testable – тестируемая, имеет критерии приемки.

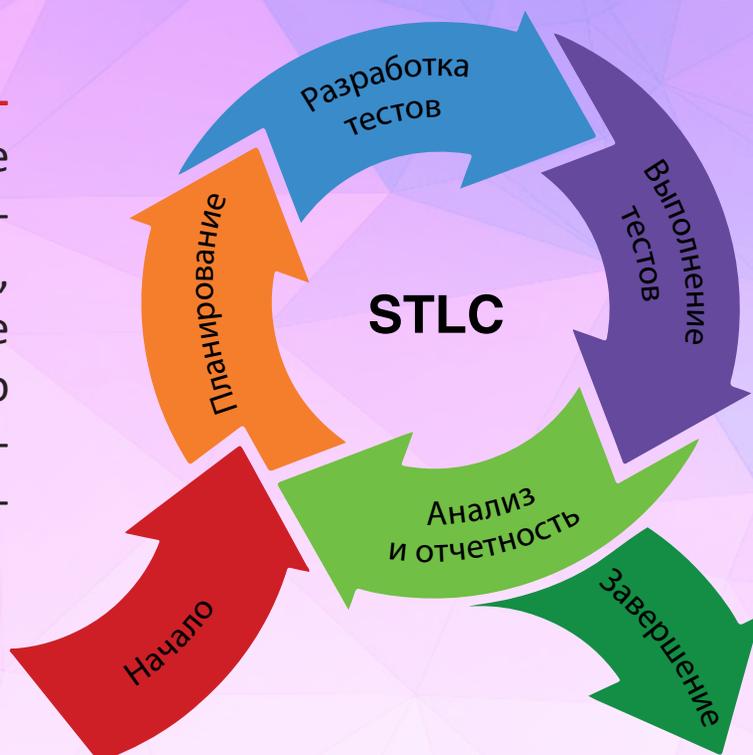


Бэклог продукта (*product backlog*) – это упорядоченный набор элементов, очередь задач, перечень всех функций, которые заинтересованные люди хотят получить от продукта. Этот список содержит краткие описания всех желаемых возможностей продукта.

Бэклог спринта – набор элементов из бэклога продукта для исполнения в текущем спринте.

ЖИЗНЕННЫЙ ЦИКЛ ТЕСТИРОВАНИЯ (STLC)

Жизненный цикл тестирования ПО (Software testing life cycle) – это последовательность действий, проводимых в процессе тестирования, с помощью которых гарантируется качество ПО и его соответствие требованиям.



ЭТАПЫ STLC-ЦИКЛА

- **Общее планирование и анализ требований.** Необходимо получить ответы на следующие вопросы: что нам предстоит тестировать; как много будет работы; какие есть сложности; всё ли необходимое у нас есть и т.п.
- **Уточнение критериев приёмки.** Позволяет сформулировать метрики и признаки необходимости начала (приостановки/возобновления/завершения) тестирования.
- **Уточнение стратегии тестирования.** Рассматриваются и уточняются те части стратегии тестирования, которые актуальны для текущей итерации.
- **Разработка тест-кейсов.** Посвящена разработке, пересмотру и уточнению тест-кейсов.
- **Выполнение тест-кейсов и фиксация найденных дефектов.** Дефекты исправляются сразу по факту их обнаружения в процессе выполнения тест-кейсов.
- **Анализ результатов тестирования и отчётность.** Создается отчет о результатах тестирования. QA-команда обсуждает и анализирует баги, делает выводы из возникших проблем, чтобы избежать подобных проблем в будущем.



Следует начинать тестирование на ранних стадиях жизненного цикла разработки ПО, чтобы найти дефекты как можно раньше.

Принцип №3 – Раннее тестирование (Early testing).

ВИДЫ ТЕСТИРОВАНИЯ

упрощенная классификация

ПО ЗАПУСКУ КОДА НА ИСПОЛНЕНИЕ:

- **Статическое тестирование** – без запуска.
- **Динамическое тестирование** – с запуском.

ПО ДОСТУПУ К КОДУ И АРХИТЕКТУРЕ ПРИЛОЖЕНИЯ:

- **Метод белого ящика** – доступ к коду есть.
- **Метод чёрного ящика** – доступа к коду нет.
- **Метод серого ящика** – к части кода доступ есть, к части – нет.

ПО СТЕПЕНИ АВТОМАТИЗАЦИИ:

- Ручное тестирование.
- Автоматизированное тестирование.

ПО УРОВНЮ ДЕТАЛИЗАЦИИ ПРИЛОЖЕНИЯ:

- **Модульное тестирование**.
- **Интеграционное тестирование** – проверка взаимодействия между модулями.
- **Системное тестирование** – приложение проверяется как единое целое.

ПО СТЕПЕНИ ВАЖНОСТИ Тестируемых функций:

- **Дымовое тестирование** – проверка самой важной, самой ключевой функциональности приложения.
- **Тестирование критического пути** – проверка функциональности, используемой типичными пользователями в типичной повседневной деятельности.
- **Расширенное тестирование** – проверка всей (остальной) функциональности, заявленной в требованиях.

ПО ПРИНЦИПАМ РАБОТЫ С ПРИЛОЖЕНИЕМ:

- **Позитивное тестирование** – это тестирование с применением сценариев, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. С его помощью мы можем определить, что система делает то, для чего и была создана.
- **Негативное тестирование** – тестирование направлено на проверку устойчивости системы к различным воздействиям, валидации неверных данных, обработку исключительных ситуаций.

ТЕСТИРОВАНИЕ ПО ЗАПУСКУ КОДА НА ИСПОЛНЕНИЕ

СТАТИЧЕСКОЕ ТЕСТИРОВАНИЕ – при статическом тестировании код не выполняется (программа не запускается).

Проверяется:

- **Документы** (требования, тест-кейсы, описания архитектуры приложения, схемы баз данных и т.д.).
- **Графические прототипы** (дизайн).
- **Код приложения** (выполняется программистами в рамках аудита кода (code review), являющегося специфической вариацией взаимного просмотра в применении к исходному коду).
- **Параметры** (настройки) среды исполнения приложения.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации.

ДИНАМИЧЕСКОЕ ТЕСТИРОВАНИЕ – тестирование с запуском кода на исполнение, как всего приложения целиком (системное тестирование), так нескольких взаимосвязанных частей (интеграционное тестирование), отдельных частей (модульное тестирование) и даже отдельные участки кода.

Динамическое тестирование включает в себя тестирование ПО в режиме реального времени путем предоставления входных данных и изучения результата поведения ПО. Проверка осуществляется с помощью заранее подготовленного набора тестов. Является частью процесса валидации программного обеспечения.



ВЕРИФИКАЦИЯ – процесс оценки системы или её компонентов с целью определения **удовлетворяют ли результаты** текущего этапа разработки **сформированным** в начале этапа **условиям**.

ВАЛИДАЦИЯ – проверка соответствия ПО требованиям пользователя. Программа может на 100 % соответствовать спецификации, но при этом выполнять совершенно не то, что хотел пользователь/заказчик.

ПРИМЕР. Релиз – ОСЁЛ. Заказчик по ТЗ хотел: непарнокопытное животное с седлом, 4 ногами и умеющее скакать. Верификация пройдена. Валидация – нет. Заказчик хотел коня.



ТЕСТИРОВАНИЕ ДОКУМЕНТАЦИИ И ТРЕБОВАНИЙ

Требования к ПО – это спецификация того, что должно быть реализовано. В них описано поведение системы, свойства системы или ее атрибуты. Требования к ПО состоят из трех уровней:

- **Бизнес-требования.** Цели, которые организация намерена достичь с ее помощью разрабатываемого продукта.
- **Пользовательские требования.** Цели и задачи, которые пользователи должны иметь возможность выполнять с помощью ПО.
- **Функциональные требования** определяют, каким должно быть поведение продукта в тех или иных условиях.

ЦЕЛЬ ТЕСТИРОВАНИЯ – устранить возможные ошибки на начальных этапах проектирования системы, снизить итоговую стоимость и улучшить качество продукта.

Свойства хорошего документа

- **Требования должны быть полными,** правильно и в полной мере описывать функцию, которую необходимо реализовать.
- **Однозначность.** Одинаковое восприятие документа всей командой.
- **Непротиворечивость.** Не должно быть противоречивых требований, конфликтующих между собой.
- **Необходимость.** Требования должны отражать функциональности, действительно необходимые для пользователя, для удовлетворения пользователей.
- **Выполнимость.** Насколько требования возможно реализовать.
- **Тестируемость.** Возможность проверить все прописанные требования после их реализации. Улучшить качество продукта.
- **Модифицируемость.** Это свойство характеризует простоту внесения изменений в отдельные требования и в набор требований.
- **Атомарность.** Требование описывает только одну ситуацию.
- **Прослеживаемость.**
- **Актуальность.**

Когда нет продуманного и четкого технического задания



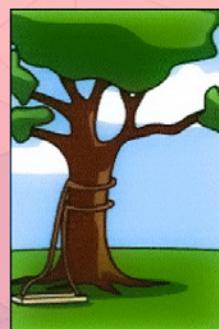
Как
объяснил
заказчик



Как понял
руководитель
проекта



Как
спроектировал
дизайнер



Как
реализовал
программист



Что реально
хотел
заказчик

ТЕСТИРОВАНИЕ ПО ДОСТУПУ К КОДУ

МЕТОД БЕЛОГО ЯЩИКА у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

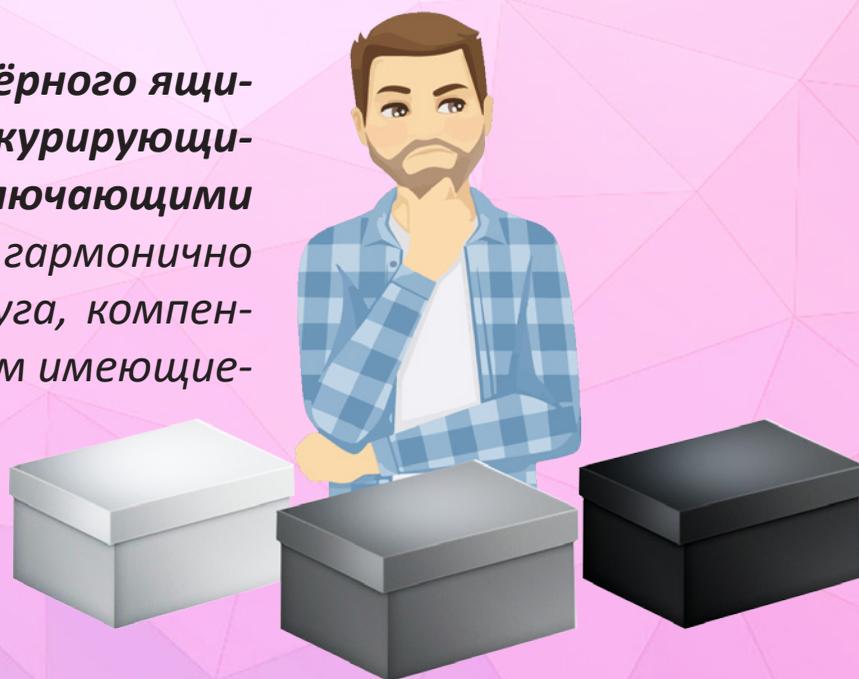
Некоторые авторы склонны связывать этот метод со статическим тестированием, но ничто не мешает тестировщику запустить код на выполнение и при этом периодически обращаться к самому коду. А модульное тестирование и вовсе предполагает запуск кода на исполнение и при этом работу именно с кодом, а не с «приложением целиком».

МЕТОД ЧЁРНОГО ЯЩИКА у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования.

Большинство видов тестирования работают по методу чёрного ящика, идею которого в альтернативном определении можно сформулировать так: тестировщик оказывает на приложение воздействия (и проверяет реакцию) тем же способом, каким при реальной эксплуатации приложения на него воздействовали бы пользователи или другие приложения.

В рамках тестирования по методу чёрного ящика основной информацией для создания тест-кейсов является документация.

Методы белого и чёрного ящика не являются конкурирующими или взаимоисключающими — напротив, они гармонично дополняют друг друга, компенсируя таким образом имеющиеся недостатки.



МЕТОД СЕРОГО ЯЩИКА комбинация методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуры у тестировщика доступ есть, а к части — нет.

КЛАССИФИКАЦИЯ ПО СТЕПЕНИ АВТОМАТИЗАЦИИ

РУЧНОЕ (МАНУАЛЬНОЕ) ТЕСТИРОВАНИЕ – это тестирование без помощи каких-либо программ, автоматизирующих работу. Строится на методах тестирования – сюда относятся и техники тест-дизайна, и техники, основанные на опыте.

Плюсы ручного тестирования:

- **Пользовательский фидбек.** Весь отчёт тестировщика может быть рассмотрен как обратная связь от потенциального пользователя.
- **UI-фидбек.** Пользовательский интерфейс полностью протестировать можно только вручную.
- **Дешевизна.** В краткосрочной перспективе ручное тестирование дешевле автоматизированной проверки.
- **Тестирование в реальном времени.** Незначительные изменения могут быть исследованы сразу, без написания кода и его исполнения.
- **Возможность исследовательского тестирования.**

Минусы ручного тестирования:

- **Человеческий фактор.** Некоторые ошибки могут остаться незамеченными.
- **Трудоемкость.**
- **Невозможность проведения некоторых видов тестов, например, нагрузочного тестирования.**



Умирает ли ручное тестирование?

В обозримом будущем создать программный компонент, который сможет идеально повторить систему человеческого восприятия, а тем более особенностей психики – вероятнее всего, невозможно.

А значит, продукт, выпускаемый под человека, должен проверяться исключительно человеком. В большей степени это применимо к визуальному оформлению продукта – «съехавшие» блоки и элементы, проблемы с верным отображением медийных файлов и прочее – всё это ещё очень долгое время будет контролироваться исключительно тестировщиком.

Автоматизация такого процесса как оценка удобства использования программ или компонентов пока что тоже невозможна.

АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ – метод тестирования ПО с использованием специальных программных средств.

Преимущества:

- Скорость выполнения выше возможностей человека.
- Автоматически формируемые и сохраняемые отчеты о результатах тестирования.
- Выполнение в фоне.
- Способность выполнить проверки непосильные для человека.

Недостатки:

- Высокие затраты.
- Автоматизация требует более тщательного планирования и управления рисками.
- В случае изменения требований, смены технологического домена, переработки интерфейсов тест-кейсы требуют создания заново.

Не подходят для автоматизации:

- Новые тестовые примеры, которые не выполнялись вручную хотя бы один раз.
- Сценарии тестирования, требования к которым часто меняются.
- Тестовые примеры, которые выполняются на разовой основе.

Выбор инструмента для автоматизации

Определение функциональности для автоматизации

Планирование, тест-дизайн и разработка

Выполнение тестов

Поддержка



Автоматизация тестирования – это деятельность, направленная на уменьшение «ручного труда» в процессе тестирования.

То есть под автоматизацией понимают применение специальных программ, которые берут на себя рутинную работу по тестированию.

КЛАССИФИКАЦИЯ ПО УРОВНЮ ДЕТАЛИЗАЦИИ ПРИЛОЖЕНИЯ



Компонентное (Модульное, Unit) тестирование – тестирование отдельных компонентов ПО.

Компонент – наименьший элемент ПО, который может быть протестирован отдельно.

Ряд авторов отдельно выделяют **Unit-тестирование** – как компонентное тестирование, но методом белого ящика, которое чаще всего выполняется разработчиками.

Интеграционное тестирование – фаза тестирования ПО, при которой отдельные модули объединяются и тестируются в группе.

Системное тестирование – тестирование, выполняемое на полной, интегрированной системе на предмет соответствия всей системы исходным требованиям. Это тестирование чёрного ящика.

Альфа-тестирование – тестирование внутри компании специальной командой тестировщиков. **Бета-тестирование** – тестирование независимыми тестерами (потребителями/юзерами).

Приёмочное тестирование – это последнее тестирование перед развертыванием, делается для проверки готовности ПО выполнять задачи и функции, поставленные при разработке.



Сквозное тестирование (end-to-end, E2E) – тщательное тестирование ПО командой тестировщиков на основании сценариев «от начала до конца» со всеми задействованными функциями. То есть тестируется рабочий процесс. Часто используется как синоним системному тестированию.

Тестирование критического пути (стр. 25) – тестирование, во время которого проверяется основная функциональность ПО, критичная для конечного пользователя при стандартном его использовании. Тестирование критического пути – это частный случай end-to-end тестирования.

КЛАССИФИКАЦИЯ ПО СТЕПЕНИ ВАЖНОСТИ ТЕСТИРУЕМЫХ ФУНКЦИЙ

ДЫМОВОЕ ТЕСТИРОВАНИЕ (smoke testing) – это тестирование, направленное на проверку самой ключевой функциональности, неработоспособность которой делает бессмысленной саму идею использования приложения.

Преимущества:

- Помогает в поиске ошибок на ранней стадии тестирования.
- Помогает в поиске проблем, которые можно получить при интеграции компонентов.
- Требуется небольшое количество тестовых случаев.

Недостатки:

- Это не детальное тестирование, в котором из-за ограниченного количества тестов мы не можем найти все важные проблемы.
- Не выполняется с негативными сценариями и с некорректными данными.

ТЕСТИРОВАНИЕ КРИТИЧЕСКОГО ПУТИ (critical path test) – направлено на проверку функциональности, используемой обычными пользователями во время их повседневной деятельности.

Тест критического пути является одним из самых распространенных видов функционального тестирования. Частота данного тестирования обусловлена, в первую очередь, необходимостью периодической проверки всего приложения в сжатые сроки.



Есть что потестить?

РАСШИРЕННОЕ ТЕСТИРОВАНИЕ (extended test) направлено на исследование всей заявленной в требованиях функциональности.

При этом здесь также учитывается, какая функциональность является более важной, а какая – менее важной. Но при наличии достаточного количества времени и иных ресурсов **тест-кейсы** этого уровня могут затронуть даже самые низкоприоритетные требования.

ПОЗИТИВНЫЕ И НЕГАТИВНЫЕ ТЕСТ-КЕЙСЫ

Позитивное тестирование – это тестирование с применением сценариев, которые соответствуют нормальному (штатному, ожидаемому) поведению системы. С его помощью мы можем определить, что система делает то, для чего и была создана.

Негативное тестирование – это тестирование в рамках которого применяются сценарии, которые соответствуют внештатному поведению тестируемой системы. Это могут быть исключительные ситуации или неверные данные.



Сначала выполняем позитивные тесты, а потом негативные.

ФРОНТЕНД/БЭКЕНД ТЕСТИРОВАНИЕ

Фронтенд – это разработка пользовательского интерфейса и функций, которые работают на клиентской стороне веб-сайта или приложения. Это всё, что видит пользователь, открывая веб-страницу, и с чем он взаимодействует. Фронтенд обменивается с бэкендом информацией через запросы.

Бэкенд – код приложения, который исполняется не на устройстве пользователя, а на удалённом сервере.

Frontend-тестирование

- Всегда выполняется в графическом интерфейсе.
- Тестер должен быть осведомлен о требованиях бизнеса, а также об использовании инструментов сред автоматизации.
- GUI используется для выполнения тестирования.
- Не требуется хранения информации в базе данных.
- Важно проверить общую функциональность приложения.

Backend-тестирование

- Включает тестирование баз данных и бизнес-логики.
- Тестер должен иметь большой опыт работы с базами данных и концепциями языка структурированных запросов (SQL).
- Важно для проверки на наличие взаимоблокировок, повреждения данных, потери данных и т.д.
- Широко используется тестирования базы данных: SQL-тестирование и API-тестирование.



КЛАССИФИКАЦИЯ В ЗАВИСИМОСТИ ОТ ЦЕЛЕЙ ТЕСТИРОВАНИЯ

Функциональное тестирование (functional testing) – это тестирование ПО в целях проверки реализуемости функциональных требований, то есть способности ПО в определённых условиях решать задачи, нужные пользователям.

Нефункциональное тестирование (non-functional testing) – анализ атрибутов качества компонента или системы, не относящихся к функциональности, то есть проверка «как работает система».

- **Тестирование производительности** – определение работоспособности, стабильности, потребления ресурсов в условиях различных сценариев использования и нагрузок.
- **Нагрузочное тестирование** – оценка поведения системы при возрастающей нагрузке, а также для определения нагрузки, которую способны выдержать компонент или система.
- **Тестирование масштабируемости** – тестирование программного обеспечения для измерения возможностей масштабирования.
- **Объёмное тестирование** – тестирование, при котором система испытывается на больших объёмах данных.
- **Стрессовое тестирование** – вид тестирования производительности, оценивающий систему на граничных значениях рабочих нагрузок или за их пределами.
- **Инсталляционное тестирование** – тестирование, направленное на проверку успешной установки и настройки, обновления или удаления приложения.
- **Тестирование интерфейса** – проверка требований к пользовательскому интерфейсу.
- **Тестирование удобства использования** – проверка того, насколько легко конечный пользователь системы может понять и освоить интерфейс.
- **Тестирование локализации** – проверка адаптации ПО для нового места эксплуатации (например, при смене языка).
- **Тестирование безопасности** – тестирование программного продукта для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.
- **Тестирование надёжности** – тестирование способности приложения выполнять свои функции в заданных условиях на протяжении заданного времени.

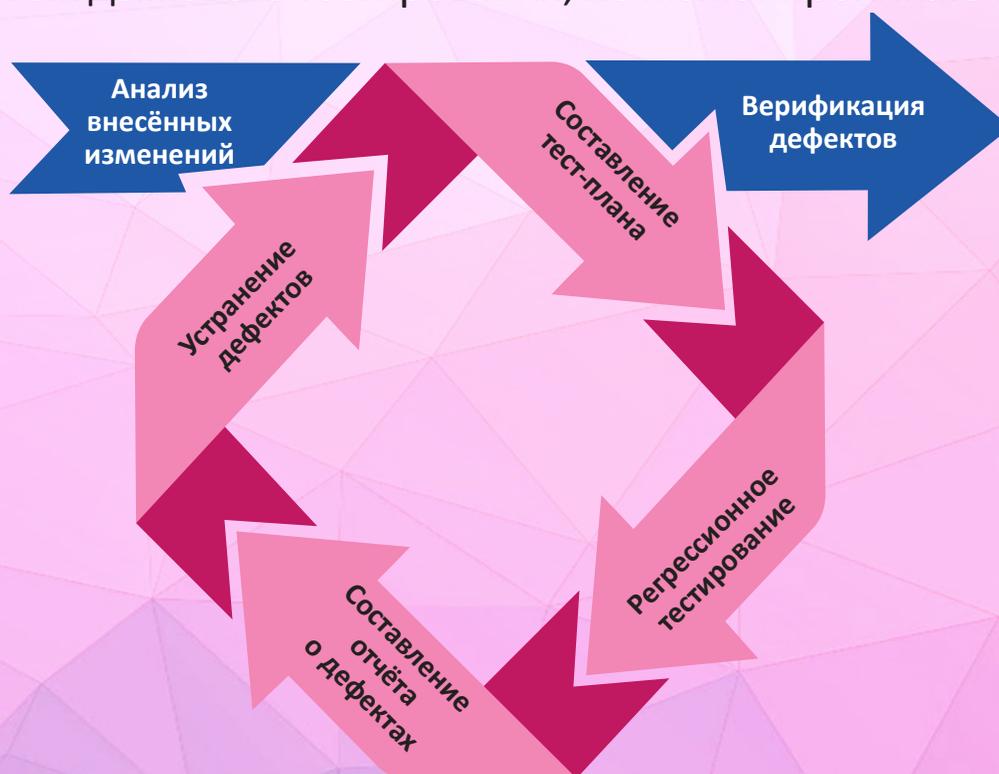
Тестирование связанное с изменениями проводится после исправления выявленных в процессе тестирования ошибок и недостатков.

Главная задача – подтвердить устранение проблемы.

- **ПОДТВЕРЖДАЮЩЕЕ ТЕСТИРОВАНИЕ** (Re-testing).
Исправлен дефект или нет.
- **РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ** (Regression Testing).
Не задело ли изменение в коде другой функционал продукта.
- **ДЫМОВОЕ ТЕСТИРОВАНИЕ** (Smoke Testing).
Проверка критически важных функций и стабильности системы в целом перед более тщательное тестированием.
- **САНИТАРНОЕ ТЕСТИРОВАНИЕ** (Sanity Testing).
Общее состояние системы в деталях. Доказательство работоспособности конкретной функции. Часто используют для проверки в результате внесенных изменений.

Санитарное тестирование ориентировано на глубинное исследование определенной функции, а **дымовое** – на тестирование большого количества функционала за самые короткие сроки.

- **ТЕСТИРОВАНИЕ СБОРКИ** (Build Verification Test).
Стабильность системы в целом, проверка критически важных функций конкретной сборки. Определение соответствия версии ПО критериям качества перед началом тестирования. Является аналогом дымового тестирования, но может проникать глубже.



Часто понятие **РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ** используют как **собирательное название** для всех видов тестирования ПО, связанных с изменениями.

ЛОКАЛИЗАЦИЯ VS ИНТЕРНАЦИОНАЛИЗАЦИЯ

Успех приложений на мировом рынке сегодня во многом зависит от того, насколько удачно приложение адаптировано для работы по всему миру.

Стратегия по подготовке приложения к интернациональному использованию называется **глобализацией**.

Тестирование локализации – это проверка содержимого приложения или сайта на соответствие лингвистическим, культурным требованиям, а также специфике конкретной страны или региона. Тестирование локализации — один из видов контроля качества, который проводится во время разработки продукта.

Тестирование интернационализации – это процесс проверки тестируемого приложения на работоспособность в разных регионах и культурах. Основная цель интернационализации — проверить, может ли код обрабатывать всю международную поддержку, не нарушая функциональность, которая может привести к потере данных или проблемам целостности данных.

Интернационализация

- ▶ UTF - кодировки
- ▶ Форматы данных
- ▶ Направление текста
- ▶ Выделение локализованных элементов из кода

Локализация

- ▶ Переводы
- ▶ Правовые требования
- ▶ Валюта и валютные операции
- ▶ Цветовые решения и символика
- ▶ Раскладки клавиатуры и горячие клавиши
- ▶ Интеграция со сторонними ресурсами



ЮЗАБИЛИТИ-ТЕСТИРОВАНИЕ

Юзабилити-тестирование – это метод оценки интерфейса со стороны удобства и эффективности его использования.

Обычных пользователей из целевой аудитории приглашают поработать с продуктом и наблюдают, как они выполняют задания и с какими сложностями сталкиваются в процессе.

Тестирование включает в себя:

- Проходимость пользовательских сценариев.
- Востребованность и впечатления от системы.
- Понятность навигации (*Насколько просто найти нужный предмет или функцию в меню или каталоге*).
- Полноту и доступность контента.
- Восприятие дизайна.



Тестирование GUI – тестирования графического пользовательского интерфейса. **Тестирование GUI и тестирование юзабилити – разные понятия.**

Юзабилити – это свойство продукта удовлетворить потребности пользователя, а **графический интерфейс** – лишь одна из составляющих юзабилити.

ТЕСТИРОВАНИЕ ДОСТУПНОСТИ

Тестирование доступности (Accessibility Testing) – это проверка ПО на пригодность к использованию людьми с нарушениями слуха, зрения, двигательной активности, т.е. это часть Usability Testing.

Вспомогательные технологии:

- Программы для распознавания речи.
- Скринридеры озвучивают текст, появляющийся на экране.
- Программы-лупы позволяют увеличить изображение на экране.
- Специальные клавиатуры для облегчения ввода текста пользователями с ограниченными двигательными возможностями.

Создаваемое ПО в идеале должно быть доступным, в нём должно учитываться использование вспомогательных технологий. Но многие вещи должны быть предусмотрены и в самом ПО.



Accessibility – это, прежде всего, доступность. Доступность нужна не только людям с ограниченными возможностями. Например, в шумной обстановке или если нельзя шуметь, а под рукой нет наушников – можно посмотреть видео с субтитрами. Или наоборот: когда нет возможности что-то прочитать, это можно прослушать.

ТЕСТОВАЯ ДОКУМЕНТАЦИЯ

Тестовая документация – это набор документов, который создается на протяжении всего цикла тестирования. Документация помогает команде однозначно трактовать шаги, сроки тестирования, результаты, обращаться к этой информации в спорных моментах.

ВИДЫ ТЕСТОВОЙ ДОКУМЕНТАЦИИ

- **Тест-план.** *Описывает весь объем работ по тестированию.*
- **Тестовая стратегия.** *Определяет то, как тестируем ПО. Это набор идей, которые направляют процесс тестирования.*
- **Чек-лист.** *Список, содержащий ряд необходимых проверок.*
- **Тест-кейс.** *Описывает наши тесты. Говорит, как их выполнить, при каких условиях и что должно получиться после выполнения шагов, которые заложены в тест-кейсе.*
- **Тест-сюит.** *Комплект тест-кейсов для исследуемого компонента или системы.*
- **Баг-репорт.** *Содержит полное описание дефекта.*
- **Отчёт о результатах тестирования.** *Здесь обобщаются все результаты работ по тестированию.*
- **Use Case.** *Сценарий взаимодействия пользователя с программным продуктом для достижения конкретной цели.*



Use Case – документ (сценарий), в котором указано, как выполнять определенную задачу.

Test Case используется для проверки работы ПО в соответствии с требованиями.

ЧЕК-ЛИСТ

Чек-лист – это список, содержащий ряд необходимых проверок во время тестирования программного продукта

ЗАДАЧИ, КОТОРЫЕ РЕШАЕТ ЧЕК-ЛИСТ:

- **Систематизирует процесс.** Чек-лист разбивает сложную работу на части и помогает не упустить из внимания важные детали
- **Облегчают делегирование.** С инструкцией сотрудникам проще разобраться в новой задаче без потери качества
- **Снижают необходимость в контроле.** Руководитель может отследить ход работ и корректировать процесс на любом этапе. Четкий алгоритм облегчает проверку задач.

ТЕСТ-КЕЙС

Тест-кейс – это документ, который описывает наши тесты. Говорит, как их выполнить, при каких условиях и что должно получиться после выполнения тех шагов, которые заложены в тест-кейсе, то есть каков ожидаемый результат.



- **ID – уникальный номер**. Обычно проставляется автоматически в системах хранения тест-кейсов.
- **Краткое описание тест-кейса** (Name). Название тест-кейса должно быть коротким и понятным. Оба эти слова важны.
- **Ссылка на требования**. Ссылка на требование или ТЗ, на основе которого был составлен тест-кейс.
- **Автор** (Author). Тестирующий, написавший тест-кейс.
- **Приоритет** (Priority) – насколько важен этот тест-кейс, в какую очередь его стоит выполнять.
- **Название/модуль/версия продукта** (Component/Version) – описание ПО, на котором можно выполнить тест-кейс.
- **Предварительные условия** (Precondition) – шаги, которые необходимо выполнить перед началом тестирования.
- **Шаги** (Steps) – точная последовательность действий для выполнения проверки. Шаги должны быть четкими и понятными.
- **Ожидаемый результат** (Expected result) – что мы получаем после выполнения шагов.
- **Приложения** (Attachments) – дополнительная информация, которая поможет выполнить тест-кейс, например, скриншоты, текстовые файлы и прочие файлы.



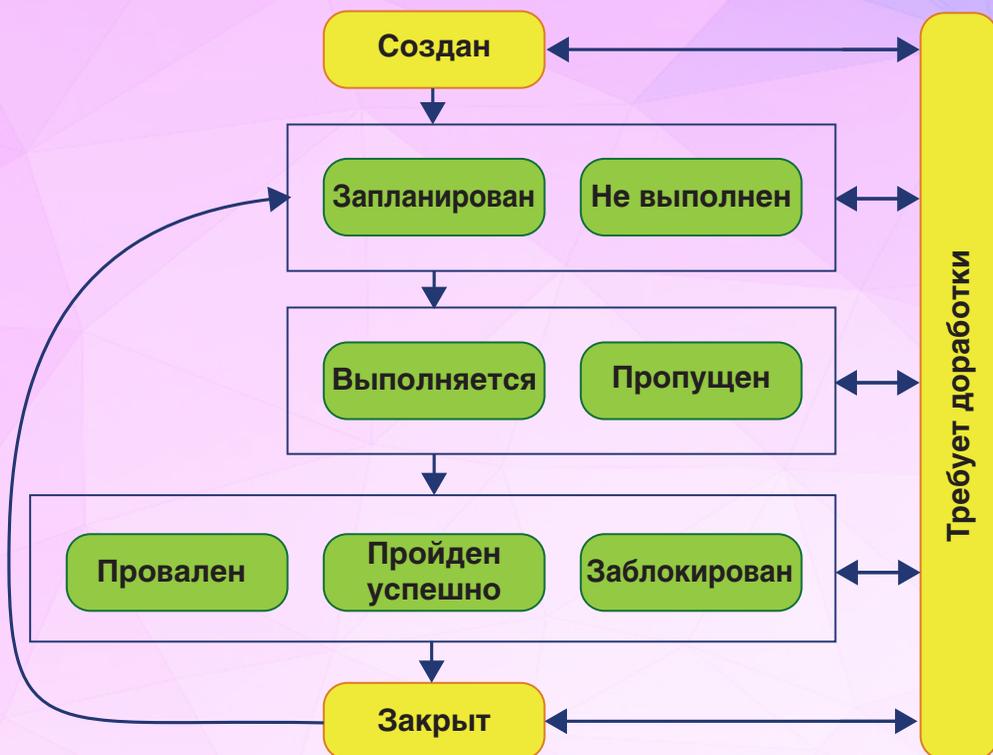
Чек-лист – это список того, что проверяем.

Тест-кейс – документ, описывающий, как проверяем.

Список тест-кейсов является своего рода чек-листом (если смотреть просто на названия тест-кейсов).

ЖИЗНЕННЫЙ ЦИКЛ ТЕСТ-КЕЙСА

В отличие от отчёта о дефекте, у которого есть полноценный развитый жизненный цикл, для тест-кейса речь идёт о наборе состояний, в которых он может находиться



- **Создан (new)** – тест-кейс автоматически переходит в это состояние после создания.
- **Запланирован (planned, ready for testing)** – состояния тест-кейса, когда он включён в план ближайшей итерации тестирования или готов для выполнения.
- **Не выполнен (not tested)** – тест-кейс готов к выполнению, но ещё не был выполнен.
- **Выполняется (work in progress)** – работа идёт, и скоро можно ожидать её результатов. Если выполнение тест-кейса занимает мало времени, это состояние, как правило, пропускается, а тест-кейс сразу переводится в состояние – «провален»/«пройден успешно»/«заблокирован».
- **Пропущен (skipped)** – выполнение тест-кейса отменяется.
- **Провален (failed)** – был обнаружен дефект.
- **Пройден успешно (passed)** – не было обнаружено дефектов, связанных с расхождением ожидаемых и фактических результатов.
- **Заблокирован (blocked)** – выполнение тест-кейса невозможно.
- **Закрыт (closed)** – на данной итерации тестирования все действия с тест-кейсом завершены.
- **Требуется доработки (not ready)** – как видно из схемы, в это состояние (или из него) тест-кейс может быть переведён в любой момент времени, если в нём будет обнаружена ошибка, если изменятся требования, по которым он был написан.

ОЦЕНКА ТРУДОЗАТРАТ

Что оцениваем:

- Человеческие умения: знания и опыт членов команды (сильно влияют на оценку).
- Ресурсы: людские, технические, и т.д.
- Время
- Стоимость: бюджет.

Кто может сделать оценку?

- Тест-аналитик
- Тестировщик (знает архитектуру системы, имеет представление о сути доработки. Плюсы/минусы/время на оценку)

ТЕСТ-ПЛАН

Тест-план (Test Plan) – документ, описывающий весь объём работ по тестированию: описания объекта тестирования, стратегии, критериев начала и окончания тестирования, необходимое оборудование и знания, оценки рисков с вариантами их разрешения.

Тест-план призван ответить на следующие вопросы:

- Что **НАДО** тестировать?
- Что **БУДЕМ** тестировать? (Тест-Аналитик).
- **КАК** будем тестировать? (Тест-Дизайнер).
- На каких уровнях будем проводить тестирование?
- Какие виды тестирования применим?
- Каким образом будем тестировать – руками или автотестами?
- **КОГДА** будем тестировать? Оценка трудозатрат и сроков.
- Какие **РИСКИ** возможны? Какие затраты времени, средств и труда они могут повлечь. Степень их влияния на исход проекта, прописать мероприятия по нейтрализации последствий срабатывания рисков.



Нет четкого шаблона, по которому необходимо писать тест-план. Главное только то, что он должен выполнять свою задачу. А именно, описать весь объём работ по тестированию и быть понятным и читабельным.

ТЕСТ-РЕПОРТ. МЕТРИКИ. ПМИ

Тест-репорт (Test report) – отчет о выполнении тест-кейсов, в нём обычно отмечается общая статистика, количество выполненных тест-кейсов и количество найденных ошибок.

Список данных, которые стоит указывать в отчете

- Состав команды
- Сроки выполнения, за которые составляется отчёт
- Описание процессов тестирования
- Изменения тестовой модели, дополнение ТК
- Процент пройденных ТК
- Критичные и блокирующие проблемы и принятые меры по их устранению
- Результаты регресса
- План на следующую итерацию/неделю/месяц

Метрика тестирования ПО определяется как количественная мера, которая помогает оценить прогресс, качество и работоспособность процесса тестирования программного обеспечения. Метрика определяет в количественном выражении степени, в которой система, системный компонент или процесс обладает заданным атрибутом.

Программа и методика испытаний (ПМИ) – это технический документ, который формализует этап тестирования продукции и составляется на автоматизированную программу (АСУ) или систему. Документ предназначен для выявления параметров, которые обеспечивают определение причин сбоя, показателей качества системы, ее соответствие различным требованиям, проверку и получение проектных решений, а также характеризуют продолжительность и период испытаний.



На практике **тестовая документация** называется ещё тестовыми **артефактами** или **артефактами тестирования**.

Артефакт – это что-то, созданное или используемое набором тестов.

НАПРИМЕР – Файл лога является артефактом.

Если ваши тесты создают **временные файлы**, это артефакты.

Если ваш тест загружает **изображения**, это артефакты.

Артефактом может быть **строка, добавленная в базу данных**.

ОТЧЁТ О ДЕФЕКТАХ (БАГ-РЕПОРТ)

БАГ РЕПОРТ (Bug Report – «отчёт об ошибке») – это технический документ, поэтому он создается по определенным правилам. Формат баг-репорта меняется в зависимости от компании, но костяк и суть всегда сохраняются. (*Что такое баг, стр. 7*)

Атрибуты баг-репорта:

1. **Заголовок ошибки** отвечает на три вопроса:

- Что произошло?
- Где появилась ошибка?
- Когда или при каких условиях найден дефект?

2. **Описание ошибки**

3. **Номер версии** (Version)

4. **Автор** баг репорта (обычно это Тестировщик)

5. **Серьёзность** (Severity)

6. **Приоритет** (Priority)

7. **Начальные условия.** В случае, если есть специфичные действия или шаги воспроизведения достаточно объёмные, то указываются начальные условия.

8. **Шаги воспроизведения.** Шаги, при которых повторяется найденная ошибка

9. **Ожидаемый результат**

10. **Фактический результат**

11. **Вложения**



ПРИОРИТЕТ И СЕРЬЕЗНОСТЬ В БАГ РЕПОРТЕ

СЕРЬЕЗНОСТЬ (Severity) – атрибут, характеризующий влияние дефекта на работоспособность ПО. Проставляется специалистом по тестированию.

ПРИОРИТЕТ (Priority) – это атрибут, указывающий на очередность выполнения задачи или устранения дефекта. Проставляется руководителем или менеджером проекта.

Приоритет отличается от Серьезности тем, что указывает, когда необходимо исправить ошибку.

ПРИМЕР бага с низкой серьёзностью, но высоким приоритетом:
Опечатка на главной странице бизнес-портала.
Системе не вредит, а репутации – да.

ТЕСТ-ДИЗАЙН

Тест-дизайн – это этап процесса тестирования ПО, на котором проектируются и создаются тестовые случаи (тест-кейсы) в соответствии с определёнными ранее критериями качества и целями тестирования.

КЛАССЫ ЭКВИВАЛЕНТНОСТИ И ГРАНИЧНЫЕ ЗНАЧЕНИЯ

КЛАССЫ ЭКВИВАЛЕНТНОСТИ – это техника, при которой мы разделяем функционал (часто диапазон возможных вводимых значений) на группы эквивалентных по своему влиянию на систему значений.



Любой тест, выполненный из одного и того же класса эквивалентности, приведет к точно такому же результату, как и выполнение всех остальных тестов из этого же класса.

ГРАНИЧНОЕ ЗНАЧЕНИЕ – это значение, которое находится на границе классов эквивалентности.

Техника анализа граничных значений – техника, проверяющая поведение системы либо отдельного модуля на граничных значениях входных данных.

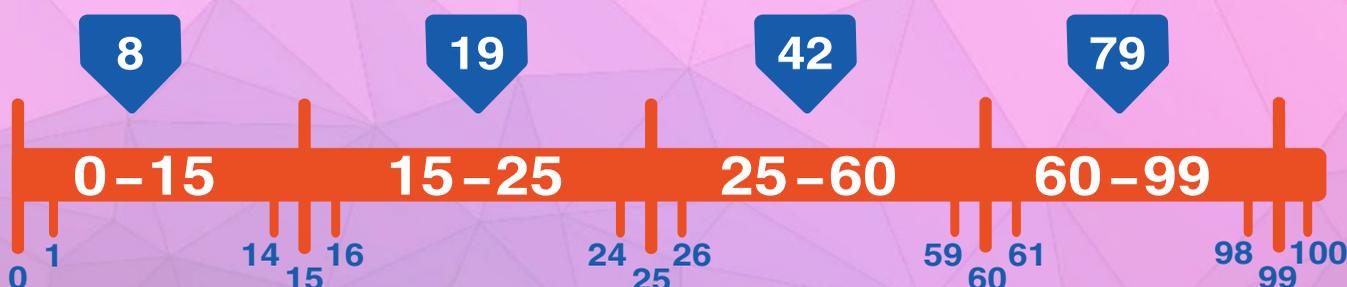


Большинство ошибок возникает именно на границах между классами эквивалентности. Т.е. тестировщику, в первую очередь, важно проверить переходы на стыке границ каждого класса.

ПРИМЕР. На картинке ниже представлено четыре возрастные группы: с рождения до 15 лет, от 15 до 25 лет, от 25 до 60 лет и от 60 до 99 лет.

Сверху отображено эквивалентное разбиение (проверки: 8 лет, 19, 42 и 79 – можно выбрать любые другие возраста в пределах группы).

Снизу – граничные значения.



СОСТОЯНИЯ И ПЕРЕХОДЫ

Тестирование состояний и переходов применяется к объекту, который может **менять свое состояние** в зависимости от **действий**, которые с ним совершают.

Тестирование перехода между состояниями помогает **анализировать поведение приложения** для различных входных условий. Тестеры могут предоставлять положительные и отрицательные входные тестовые значения и записывать поведение системы.

Этапы тестирования:

- Исследовать компонент и собрать информацию.
- Составить диаграмму и описание модели компонента.
- Сделать таблицу и скомбинировать возможные состояния и действия.

ПРИМЕР

Состояния воды:

1. Лёд – твердое состояние.
2. Вода – жидкое состояние.
3. Пар – газообразное состояние.

Состояние **ВОДА**

→ Действие **ОХЛАДИТЬ**

→ Состояние **ЛЕД**

ТАБЛИЦЫ РЕШЕНИЙ

ТАБЛИЦА РЕШЕНИЙ (Decision Table) – техника, помогающая наглядно изобразить комбинаторику условий из ТЗ.

По горизонтали. **УСЛОВИЯ**, которые влияют на результат. Обязательно в виде вопроса. **ДЕЙСТВИЕ** или следствия (описание ожидаемого результата). Обязательно утверждение и только одно.

По вертикали. **ПРАВИЛА** – комбинация входных условий, или проще **тесты**.

Когда использовать:

- Когда есть проблемы с документацией.
- В условиях постоянно меняющихся требований.

ПРИМЕР. Применение скидки в корзине покупателя

	Тест 1	Тест 2	Тест 3	Тест 4	Тест 5	Тест 6	Тест 7	Тест 8	Тест 9	Тест 10
Покупатель авторизован?	+	+	+	+	+	-	-	-	-	-
Количество товара от 1 шт. до 2 шт.?	+	-	-	-	-	+	-	-	-	-
Количество товара от 3 шт. до 4 шт.?	-	+	-	-	-	-	+	-	-	-
Количество товара от 5 шт. до 9 шт.?	-	-	+	-	-	-	-	+	-	-
Скидка за авторизацию 7%	+	+	+	+	+	-	-	-	-	-
Скидка за количество (1-2 шт) - 0%	+					+				
Скидка за количество (3-4 шт) - 3%		+					+			
Скидка за количество (5-9 шт) - 5%			+					+		

ПОПАРНОЕ ТЕСТИРОВАНИЕ

Метод попарного тестирования (Pairwise testing) основан на идее: подавляющее большинство багов выявляется тестом, проверяющим либо один параметр, либо сочетание двух. *Ошибки, причиной которых явились комбинации трех и более параметров, как правило, значительно менее критичны.*

ПРИМЕР. Имеем систему, которая зависит от нескольких входных параметров (10 параметров по два значения Вкл/Выкл). Все возможные варианты сочетания этих параметров – 1024 комбинации. Используя метод попарного тестирования – мы не тестируем все возможные сочетания, а составляем тесты так, чтобы каждое значение параметра хотя бы один раз сочеталось с каждым значением остальных тестируемых параметров.

Таким образом, метод существенно сокращает количество тестов, а значит, и время тестирования.

ПО для автоматического формирования проверок попарного тестирования: allpairs, PICT, Pairwise online tool, VPTag, ACTS и др.

ТЕХНИКА ПРЕДПОЛОЖЕНИЕ ОБ ОШИБКЕ

Предположение об ошибках – это способ предотвращения ошибок, дефектов и отказов, основанный на знаниях тестировщика, включающих:

- *Историю работы приложения в прошлом.*
- *Наиболее вероятные типы дефектов, допускаемых при разработке.*
- *Типы дефектов, которые были обнаружены в схожих приложениях*

В предугадывании ошибок нет четкой и логической схемы, которая позволила бы нам составить тест-кейсы. Т.е. нельзя сказать, что, сделав сначала Шаг №1, затем Шаг №2 и т.д., мы на выходе получим готовые проверки с максимально полным покрытием.

Наоборот, эта техника основывается на опыте тестировщика и на его умении думать креативно и деструктивно.

ИССЛЕДОВАТЕЛЬСКОЕ ТЕСТИРОВАНИЕ

Исследовательское тестирование (Exploratory Testing) – частично формализованный подход, в рамках которого **тестировщик выполняет работу с продуктом по выбранному сценарию**, который, в свою очередь, дорабатывается в процессе выполнения с целью более полного исследования приложения.

АД-НОС ТЕСТИРОВАНИЕ

Свободное (интуитивное) тестирование (Ad-hoc Testing) – полностью неформализованный подход, в котором **не предполагается использования ни тест-кейсов, ни чек-листов, ни сценариев**. Тестировщик опирается на свою интуицию и опыт для спонтанного выполнения с продуктом действий, которые, как он считает, могут обнаружить ошибку.



Исследовательское и свободное тестирование – это разные техники исследования продукта с разной степенью формализации, разными задачами.

Разница между свободным и исследовательским тестированием в том, что, теоретически, свободное может провести кто угодно. А для проведения исследовательского необходимо мастерство и владение определёнными техниками.

Можно сказать, что свободным тестированием занимаются бета-тестировщики, которые добровольно вызвались использовать продукт и сообщать об ошибках. Они как раз понятия не имеют ни о техниках тестирования, ни о его методах и принципах.

МОНКЕЙ ТЕСТИРОВАНИЕ

Monkey тестирование (Monkey Testing) – это метод тестирования ПО, при котором **тестировщик вводит любые случайные входные данные** без заранее определенных тестовых случаев и проверяет поведение программного приложения, независимо от того, даёт ли оно сбой или нет. То есть тестируем без цели, без плана. Просто тыкаемся везде с намерением что-то сломать. Как обезьянка.

МОБИЛЬНОЕ ТЕСТИРОВАНИЕ

Мобильное тестирование – процесс тестирования приложений для современных мобильных устройств на функциональность, удобство использования, производительность и др.

Особенности тестирования мобильных приложений

- *Тестирование пользовательского взаимодействия* – удобства работы с приложением: свайпы, тапы, скролы и т.п.
- *Тестирование совместимости* – установка на разные ОС, платформы, на разных моделях, проверка на разных разрешениях и т.п.
- *Тестирование подключения* – проверка на разных типах подключения (wi-fi, мобильная сеть), переключение типов и оффлайн работа.
- *Тестирование производительности* – утечка памяти, стабильность работы при большом количестве пользователей и т.п.
- *Тестирование локализации* – проверка размещения локализованного (переведённого) текста на экране, формата дат и т.д.

ТИПЫ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

- **НАТИВНЫЕ ПРИЛОЖЕНИЯ** разрабатываются «под платформу», т.е. для конкретной операционной системы: iOS, Android или Windows. Разрабатываются на языке высокого уровня и компилируется в т.н. native-код ОС, обеспечивающий максимальную производительность.

Особенности

- *Приложение устанавливается на смартфон из оф. магазина.*
- *Нативные сервисы могут работать независимо от подключения к интернету, хотя часть из них требует подключения.*

Преимущества

- *Высокая производительность.*
- *Имеют доступ к устройству пользователя и таким функциям как Bluetooth, списки контактов, камера, NFC и др.*
- *Нативные приложения более безопасны с точки зрения защиты данных пользователя.*

Недостатки

- *Стоимость разработки. Нужно разрабатывать отдельно для каждой ОС, а потом поддерживать его.*
- *Нативные приложения занимают место в памяти устройства пользователя, причем с каждым новым обновлением это занятое место может расти.*

- **ВЕБ-ПРИЛОЖЕНИЯ** представляют собой адаптированные веб-сайты, которые открываются через браузеры. Пользователь не скачивает приложение и не хранит его на своем устройстве. Если его «скачивают», скорее всего, речь идет о том, что оно добавляется в закладки браузера. Одним из самых распространенных подвидов считают PWA — прогрессивные веб-приложения, которые, по сути, являются нативными приложениями внутри браузера. Сложно выделить примеры, чтобы не ошибиться.

ПРИМЕР. Google Maps, Google Документы, Tilda и тд.

Преимущества

- *Не требуют настроек под операционную систему, что делает разработку быстрой и менее дорогой.*
- *Не требуют загрузки и не занимают место на устройстве пользователя.*
- *Не требуют обновлений, поэтому в теории их проще поддерживать со стороны разработчика и, опять же, пользователю не надо устанавливать никакие обновления, чтобы работать с площадкой.*

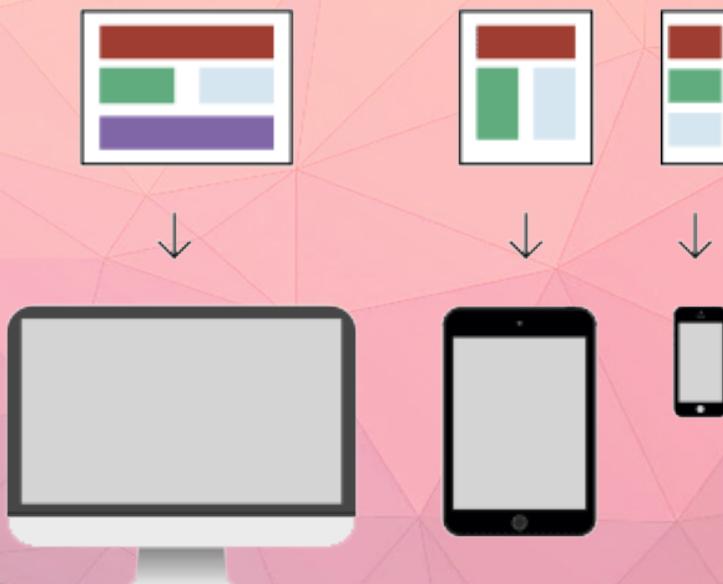
Недостатки

- *Веб-приложения зависят от браузера. И функции, которые доступны в одном браузере, могут не поддерживаться в другом. Это означает, что пользовательский опыт будет отличаться.*
- *Веб-приложения не работают без подключения к интернету.*

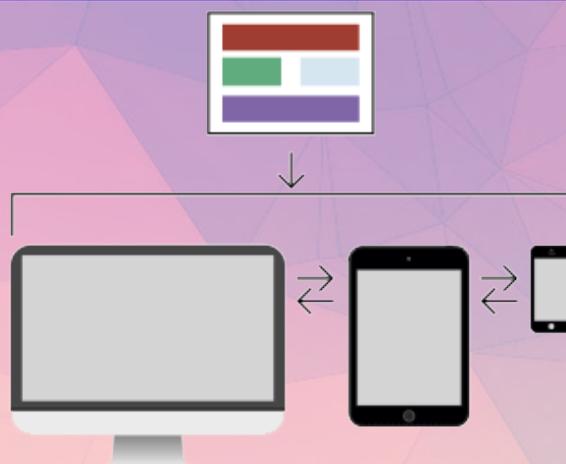


Адаптивность и Отзывчивость

АДАПТИВНЫЙ дизайн Adaptive Design (AWD) проектирование сайта с несколькими статичными макетами для разных типов устройств (мобильные устройства, планшеты, настольные компьютеры). То есть макеты загружаются при определенных размерах окна браузера устройства, базируясь на контрольных (переломных) точках.



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



PWA или Progressive Web Application

Прогрессивное веб-приложение – технология, которая позволяет установить сайт на смартфон как приложение.

Бренды Twitter, Uber, Telegram, Starbucks, AliExpress, Aviasales используют приложения на базе PWA как основное приложение либо в дополнение к мобильному приложению.

Instant Apps

Мгновенные приложения – это такие приложения, которые можно скачать и запустить без необходимости прохождения полного цикла установки. Доступны только для Android.

- **ГИБРИДНЫЕ ПРИЛОЖЕНИЯ** – приложение, которое сочетает в себе элементы нативного и веб-приложения. Приложение необходимо загружать на устройство (как нативные приложения), но написано оно с помощью HTML, CSS и JavaScript.

Плюсы

- Бюджетная и быстрая разработка по сравнению с нативными.
- Кроссплатформенность.
- Более быстрая загрузка.
- Возможность взаимодействия с ОС устройства.

Минусы

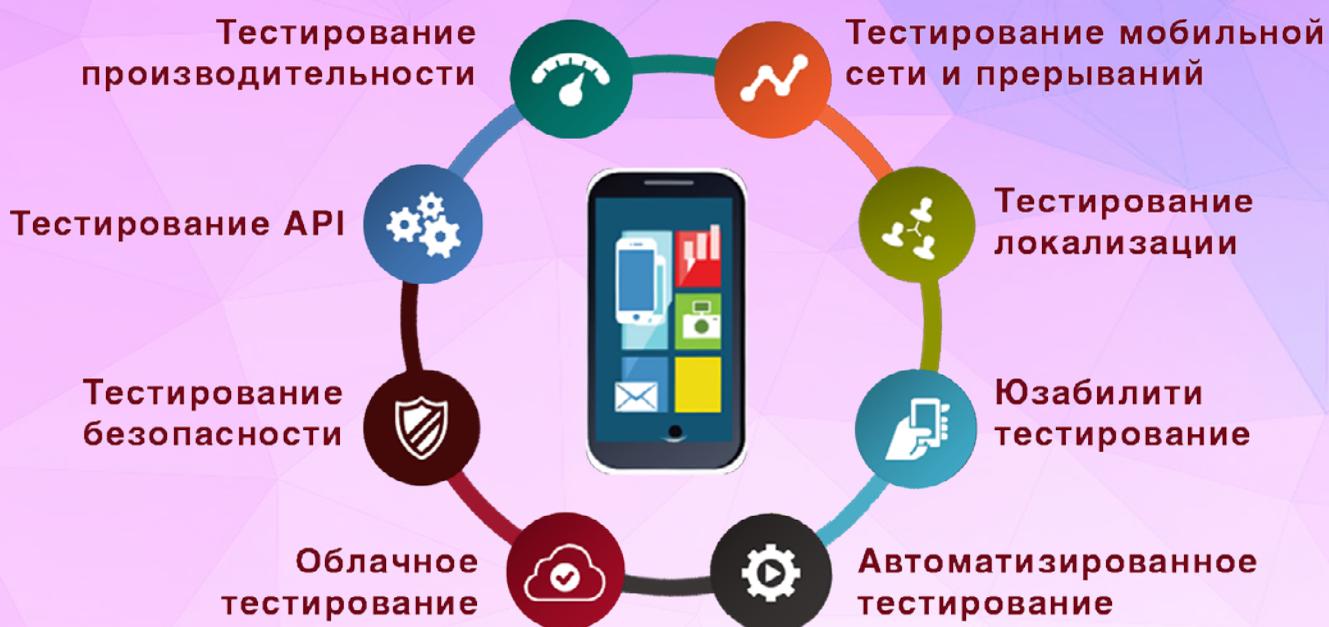
- Производительность ниже, чем у нативных приложений;
- Графика менее адаптирована к ОС в сравнении с нативным приложением.



Кроссбраузерность – это способность веб-ресурса отображаться одинаково и работать во всех популярных браузерах без перебоев в функционировании и ошибок в верстке, а также с одинаково корректной читабельностью контента.

Кроссплатформенность – способность ПО работать с несколькими аппаратными платформами или операционными системами.

ЧЕК-ЛИСТ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ



- **Функциональное тестирование**
- **Тестирование совместимости** с другими версиями ОС, различными оболочками и сторонними сервисами (например, оплата), а также аппаратным обеспечением устройства, подключение внешних устройств, тестирование прерываний.
- **Тестирование безопасности.** Тестирование разрешений (доступ к камере/микрофону/галерее/и т.д.). Тестирование передачи данных пользователя (пароли), и т.д.
- **Тестирование локализации и глобализации**
- **Тестирование удобства использования**
- **Стрессовое тестирование**
- **Кросс-платформенное тестирование**
- **Тестирование производительности**

СИМУЛЯТОРЫ И ЭМУЛЯТОРЫ

Эмулятор – имитирует программную (софт) и аппаратную (железо) часть устройства. Цель работы на эмуляторе – создание точной модели устройства и полная проверка корректности работы программы на этом устройстве.

Симулятор – имитирует ПО устройства. Цель работы на симуляторе – понять, как работать в оригинальной программе, изучить её интерфейс, а также понять, как будет реагировать программа на действия пользователей.



Эмуляторы не могут имитировать проблемы с батареей, входящие прерывания, проблемы с датчиками GPS и освещения, команды жестами, проблемы с сенсорным экраном, цветопередачу.

DevTools – это набор инструментов, встроенных в браузер, для создания и отладки сайтов. С их помощью можно просматривать исходный код сайта, отлаживать работу frontend: HTML, CSS и JavaScript. Также DevTools позволяет проверять сетевой трафик, быстродействие сайта и многое другое. С помощью режима эмуляции DevTools позволяет просматривать веб-страницы в мобильном виде.

Сниффер – это инструмент, позволяющий перехватывать, анализировать и модернизировать все запросы, которые через него проходят. С помощью него можно извлечь из потока какие-либо сведения или создать нужный ответ сервера.

Кеш (Cache) – представляет собой копии загруженных веб-страниц. Если повторно заходить на сайт, тогда его загрузка происходит не из интернета, а с жесткого диска, где хранятся временные файлы. Это ускоряет работу браузера. Веб-страницы могут отображаться некорректно в связи с тем, что в них были внесены изменения, а браузер продолжает использовать устаревшие данные из кэша.

Для обеспечения корректности отображения веб-страниц перед проведением тестирования нужно обязательно очищать кэш браузера.

Куки (Cookie) – временные файлы, хранящиеся на жестком диске компьютера пользователя. Куки хранят настройки сайтов, которые пользователь посещал. Самая распространённая функция – сохранение паролей, которая позволяет не вводить комбинацию логин + пароль каждый раз при входе на сайт.

ТЕСТИРОВАНИЕ ПРЕРЫВАНИЙ – это тестирование реакции мобильного приложения на прерывание и возвращение к своему предыдущему состоянию.

Виды прерываний:

- Низкий заряд батареи.
- Входящий звонок.
- Входящие смс.
- Входящее оповещение из другого мобильного приложения.
- Подключен для зарядки.
- Отключен от зарядки.
- Устройство выключено.
- Напоминания об обновлении приложения.
- Аварийная сигнализация.
- Потеря сетевого подключения. Восстановление сетевого подключения.

ОСНОВЫ БАЗ ДАННЫХ

База данных (БД) – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе.

Система управления базами данных (СУБД) – комплекс программ, позволяющих создать БД и манипулировать данными (вставлять, обновлять, удалять и выбирать).

ВИДЫ БАЗ ДАННЫХ:

- **Фактографическая**, содержит краткую информацию об объектах системы, формат которой строго фиксирован.
- **Документальная**, включает документы разного вида, в том числе текстовые, графические, звуковые, мультимедийные.
- **Распределенная**, БД с разными частями, которые хранятся на различных компьютерах, объединенных в сеть.
- **Централизованная**, представляет собой базу данных, местом хранения которой является один компьютер.
- **Реляционная**, имеет табличную организацию данных.
- **Неструктурированная (NoSQL)** – это нереляционный тип БД. NoSQL получили широкое распространение в связи с простотой разработки, функциональностью и производительностью при любых масштабах.

РЕЛЯЦИОННАЯ БД – это набор данных с **предопределенными связями** между ними.

Данные организованы в виде набора **таблиц**, где хранится информация об объектах. В **столбце** хранится определенный тип данных, в каждой **ячейке** – значение атрибута. **Строка** представляет собой набор связанных значений, относящихся к одному объекту или сущности.

В большинстве БД для записи и запросов данных используется язык структурированных запросов – **SQL**.



Структура данных должна быть независима от программ, использующих эти данные, так, чтобы данные можно было добавлять или перестраивать без изменения этих программ.

Независимость данных – это свойство СУБД, которое помогает вам изменять схему базы данных на одном уровне системы базы данных, не требуя изменения схемы на следующем более высоком уровне.

Примеры Базы Данных



Ключ СУБД – это атрибут или набор атрибутов, который помогает идентифицировать строку в таблице.

Первичный ключ (primary key) – поле, каждый элемент которого однозначно определяет запись таблицы. Используется для идентификации объекта.

Ключ внешний (Foreign key) – это столбец или сочетание столбцов, которое применяется для принудительного установления связи между данными одной БД.



SQL ЗАПРОСЫ

SQL (Structured Query Language) в переводе **язык структурированных запросов** – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной БД, управляемой соответствующей СУБД.

*Простыми словами **ЯЗЫК SQL** – стандартный язык управления реляционными базами данных с архитектурой клиент-сервер.*

ЦЕЛЬ РАЗРАБОТКИ – создание простого непроцедурного языка, которым мог бы воспользоваться любой пользователь, даже не имеющий навыков программирования.

ТИПЫ SQL ЗАПРОСОВ

КОМАНДЫ, РАБОТАЮЩИЕ СО СТРУКТУРОЙ БД.

- **CREATE** – создать. Например **CREATE TABLE** – создать таблицу или **CREATE USER** – создать пользователя.
- **ALTER** – модифицировать. Этот запрос используется при внесении изменений в саму БД или в ее часть.
- **DROP** – удалить. Запрос относится к БД и её частям.

КОМАНДЫ, РАБОТАЮЩИЕ С ДАННЫМИ.

- **SELECT** – выборка данных.
- **INSERT** – вставка новых данных.
- **UPDATE** – обновление данных.
- **DELETE** – удаление данных.
- **MERGE** – слияние данных.

КОМАНДЫ, РАБОТАЮЩИЕ С ПРАВАМИ ДОСТУПА.

- **GRANT** – разрешение пользователю на проведение определенных операций с БД или данными.
- **REVOKE** – отзыв выданного разрешения.
- **DENY** – установка запрета, имеющего приоритет над разрешением.

СТРУКТУРА SQL-ЗАПРОСОВ

При составлении SQL-запроса для работы с базами данных в СУБД (MySQL, Microsoft SQL Server, PostgreSQL) **вводятся следующие параметры отбора:**

- Названия таблиц, из которых необходимо извлечь данные.
- Поля, значения которых требуется вернуть к исходным после внесения изменений в БД.
- Связи между таблицами.
- Условия выборки.
- Вспомогательные критерии отбора (ограничения, способы представления информации, тип сортировки).

НАПРИМЕР, для выборки по клиентам, приносящим интернет-магазину наибольшую прибыль, для работы с БД фирмы строится запрос, имеющий вид:

SELECT col1, col2, col3 (перечисление колонок, которые нужно отобразить) **from table** (имя таблицы) **where** (указание на последующий фильтр) **clause** (критерий отбора).

ОСНОВЫ API

API (Application programming interface) – описание способов и правил, по которым одна компьютерная программа может взаимодействовать (общаться и обмениваются данными) с другой.

ВНЕШНИЕ И ВНУТРЕННИЕ API

Внешние (сторонние) API. Доступны для использования всем, включая сторонних разработчиков программного обеспечения. Примером являются гугл или яндекс карты, регистрация через соц. сети. То есть разрабатывать отдельно их не надо, это готовые решения для использования.

Внутренние API. Разрабатываемые функции под нужды конкретного сайта. Пример – поиск по сайту с фильтрацией по каким-то критериям.

HTTP ПРОТОКОЛ

HTTP (HyperText Transfer Protocol, протокол передачи гипертекста). Протокол клиент-серверного взаимодействия, он инициирует запросы (сообщения) к серверу самим получателем. Обмен сообщениями идёт по схеме «запрос-ответ».

На данный момент благодаря протоколу HTTP обеспечивается работа Всемирной паутины. Обычно приложение осуществляет доступ к веб-ресурсам через веб-браузер.

Для идентификации ресурсов HTTP использует URI.

URI – последовательность символов, идентифицирующая физический или абстрактный ресурс (не обязательно должен быть доступен через сеть Интернет). Если просто – это строка, указывающая на ресурс: HTML-страница, CSS-файл, изображение и т.д.

Самый популярный тип URI – это **URL** (Uniform Resource Locator), который также называют **веб-адресом**. Помимо идентификации ресурса, URL предоставляет и информацию о местонахождении этого ресурса в Интернете.

HTTP vs HTTPS



HTTP – это открытый протокол передачи данных между браузером и сервером. **HTTPS** – тот же HTTP, но с добавленными методами шифрования данных и проверки безопасности, т.е. **защищенный протокол передачи данных**.

HTTP ЗАПРОСЫ – это сообщения, отправляемые клиентом, чтобы инициировать реакцию со стороны сервера.

HTTP ОТВЕТЫ – это сообщения, которое посылает сервер в ответ на запрос.

СТРУКТУРА HTTP ЗАПРОСА

- **Строка запроса** (*Request line*). В строке запроса указывается **метод передачи**, версия протокола HTTP и URL, к которому должен обратиться сервер.

Метод HTTP (*HTTP Method*) – последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом.

Простыми словами МЕТОД – это определенное слово, которое мы говорим серверу, а сервер по этому слову определяет, что ему нужно делать. Обычно это короткое английское слово, записанное заглавными буквами.

GET – используется, когда мы хотим получить какие-то данные с сервера.

POST – используется, когда нам нужно создать новый объект или внести какие-либо изменения.

PUT – используется, если нам надо внести изменения в существующий объект.

DELETE – используется, когда нужно удалить объекты или сущности.

- **Заголовки** (*Request Headers*). Характеризует тело сообщения, какие параметры передаем. Между заголовком и телом есть пустая разделительная строка.
- **Тело запроса** (*Request Body*). Информация, которую передал браузер при запросе страницы. Тело сообщения не является обязательным параметром, оно присутствует, только если браузер запросил страницу методом POST (т.е. когда нужно передать данные для создания/изменения сущности).

СТРУКТУРА HTTP ОТВЕТА

- **Код состояния** (*Status code*). Часть первой строки ответа сервера, который информирует клиента о результате запроса. Состоит он из трех цифр, первая из которых указывает на класс состояния.
- **Заголовки** (*Response headers*). Служебная информация.
- **Тело сообщения** (*Response body*). Данные, которые посылает сервер в ответ на запрос.

КЛИЕНТ-СЕРВЕРНАЯ АРХИТЕКТУРА

Клиент – компьютерное устройство на стороне пользователя, которое отправляет запросы серверу (касающиеся выполнения определенных задач или предоставления конкретной информации) и принимает от него ответы.

Сервер – это специальный компьютер для хранения информации и обеспечения доступа к ней с удалённых клиентских устройств. Чаще всего на серверах хранят базы данных, различные документы, данные 1С. То есть сервер это устройство, предоставляющее ресурс – задачу или информацию.

Клиент-серверное взаимодействие – это обмен данными между клиентом и сервером.

Клиент-серверная архитектура



Клиент

Клиент отправляет HTTP запрос к API, а в ответ получает HTTP ответ и обрабатывает его.

Сервер

Клиент общается с сервером через API. Сервер принимает HTTP запрос, обрабатывает его и выдает клиенту HTTP ответ.

База данных

База данных принимает SQL запрос от сервера и выдает ответ обратно.

На клиент-серверной архитектуре построены все сайты и интернет-сервисы, а также ряд десктоп-программ, которые передают данные по интернету.

Архитектура «клиент-сервер» определяет общие принципы организации взаимодействия в сети, где имеются серверы, узлы-поставщики некоторых специфичных функций (сервисов) и клиенты (потребители этих функций). Практические реализации такой архитектуры называются клиент-серверными технологиями.

- **Двухзвенная.** Распределение трёх базовых компонентов между двумя узлами – клиентом и сервером.
- **Многоуровневая** (трехуровневая как частный случай). Суть многоуровневой архитектуры заключается в том, что запрос клиента обрабатывается сразу несколькими серверами. В качестве примера можно привести любую современную СУБД (за некоторым исключением).

ТЕСТИРОВАНИЕ API

Тестирование API – это тип тестирования, который сосредоточен на уровне бизнес-логики архитектуры ПО.

Целью тестирования API является проверка функциональности, надежности, производительности и безопасности программных интерфейсов. Тестирование API относится к интеграционному тестированию, а значит, в ходе него можно отловить ошибки взаимодействия между модулями системы или между системами. Также рассматриваются вопросы безопасности.

Веб-сервисы

– это способ связи (обмен данными) между двумя электронными устройствами (приложениями) по сети.

ПРОТОКОЛЫ РЕАЛИЗАЦИИ ВЕБ-СЕРВИСОВ:

XML-RPC (XML Remote Procedure Call) – протокол удаленного вызова процедур с использованием XML. Прародитель SOAP.

SOAP (Simple Object Access Protocol) – протокол обмена структурированными сообщениями в распределённой вычислительной среде. Протокол используется для обмена произвольными сообщениями в формате XML.

JSON-RPC (JSON Remote Procedure Call) – это текстовый формат обмена данными, данные передаются в формате JSON. (JavaScript Object Notation). Структура JSON состоит из набора пар **ключ – значения**.

REST (Representational State Transfer, передача самоописываемого состояния) – архитектурный стиль взаимодействия компьютерных систем в сети, основанный на методах протокола HTTP. Другими словами, REST – это набор правил о том, как программисту организовать написание кода серверного приложения, чтобы все системы легко обменивались данными и приложение можно было масштабировать.

Тестирование API обладает рядом преимуществ перед UI:

- Точное понимание, где происходит ошибка и чем она вызвана.
- Тратится меньше времени на подготовку тестовых данных.
- Возможно выполнение тестов на больших объемах данных с приемлемой скоростью.
- Можно начать тестирование на ранних этапах, когда еще нет интерфейса.

ВВЕДЕНИЕ В HTML И CSS



Языки **HTML** и **CSS** предназначены для вёрстки сайтов. Язык **PHP** нужен для программирования сайта, с его помощью можно, к примеру, сделать регистрацию пользователей.

Язык **JavaScript** нужен для того, чтобы «оживить» сайт: к примеру, сделать меняющиеся картинки (слайдер).

Язык HTML

Язык HTML (HyperText Markup Language) — язык гипертекстовой разметки. Базовый язык для создания веб-страниц.

Страница сайта – это обычный текстовый файл с расширением **.html**. Внутри этого файла и хранится текст HTML страницы вместе с тегами.



HTML состоит из:

тегов, атрибутов, значений атрибутов, содержимого элемента и цельного элемента.

HTML теги – это специальные команды для браузера. Они говорят, например, что следует считать заголовком страницы, а что абзацем.

Теги строятся по такому принципу: уголок **<**, потом имя тега, а потом уголок **>**. Имя тега может состоять из английских букв и цифр. Примеры тегов: **<h1>**, **<p>**, ****.

Теги обычно пишутся парами – открывающий тег и закрывающий. Разница между ними в том, что в закрывающем теге после уголка **<** стоит слеш **/**. **Пример: <p> </p>**. Все, что попадает между открывающим и закрывающим тегами, **подпадает под воздействие нашего тега**. Бывают теги, которые **не нужно закрывать**, например, **
** или ****.

АТТРИБУТЫ – специальные команды, которые расширяют действие тега. Атрибуты размещаются внутри открывающего тега в таком формате: **<тег атрибут1="значение" атрибут2="значение">**



Кавычки могут быть любыми – одинарными или двойными, допустимо вообще их не ставить, если значение атрибута состоит из одного слова (но это нежелательно).

СТРУКТУРА HTML-ДОКУМЕНТА:

Базовый набор тегов:

<html> – указывает, что это HTML код.

Перед тегом **<html>** обычно пишется конструкция **DOCTYPE** (**<!DOCTYPE html>**). **DOCTYPE** отвечает за корректное отображение веб-страницы браузером, определяет версию HTML (например, html) и соответствующий DTD-файл в Интернете.

<head> – содержит техническую информацию, например, заголовок страницы, подключаемые стили и скрипты, мета-информацию и т.д.

<title> (заголовок) – содержит заголовок HTML-документа который отображается во вкладке браузера.

<body> – содержимое документа, которое видит пользователь.

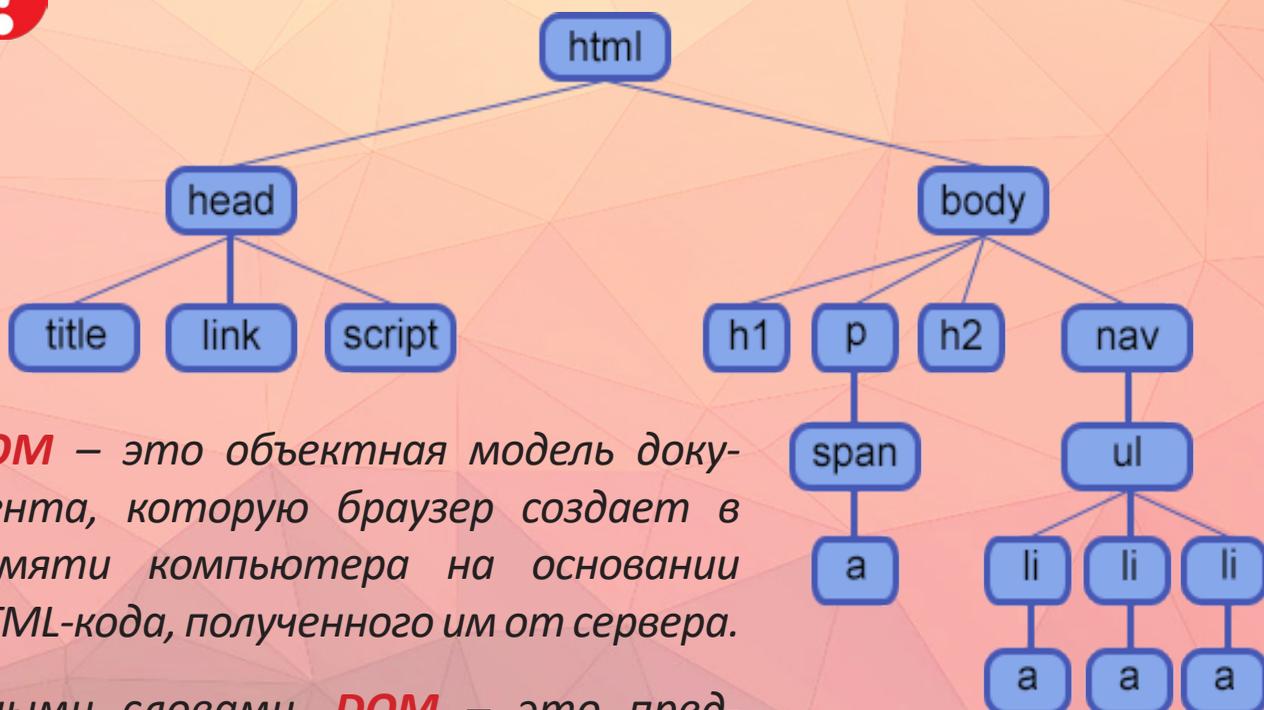
<meta> – задаёт описание содержимого страницы и ключевые слова, автора HTML-документа и др. свойства метаданных, например, кодировку (ставится в атрибуте **charset** и обычно имеет значение **utf-8**).

АБЗАЦЫ – Каждый абзац начинается с новой строки и имеет так называемую красную строку. Создается с помощью тега **<p>**

ЗАГОЛОВКИ ТЕКСТА используются для определения нового раздела или подраздела. Заголовки создаются с помощью тегов **<h1>**, **<h2>**, **<h3>**, **<h4>**, **<h5>**, **<h6>**. Они имеют разную степень важности. В заголовке **h1** следует располагать название всей HTML страницы, в **h2** – название блоков страницы, в **h3** – название подблоков и т.д. Все заголовки по умолчанию жирные и имеют разный размер.



DOM (Document Object Model)



DOM – это объектная модель документа, которую браузер создает в памяти компьютера на основании HTML-кода, полученного им от сервера.

Иными словами, **DOM** – это представление HTML-документа в виде дерева тегов.

Язык CSS

CSS расшифровывается как **Cascading Style Sheets**, что в переводе означает «Каскадные Таблицы Стилей».

ЯЗЫК CSS – это язык разметки, используемый для визуального оформления веб-сайтов, он расширяет возможности HTML.

CSS позволяет быстро изменить визуальное оформление сайта, не прибегая к использованию более сложных языков программирования. С его помощью можно поменять цвета, шрифты, фон, в общем заниматься красотой сайта.

СИНТАКСИС

Основная задача Каскадных Стилей – рассказать движку браузера, как отрисовать элементы страницы. Каким цветом, как позиционировать их на странице, как украшать и т.д.

Синтаксис CSS – это плоский список CSS-правил.

CSS-правило состоит:

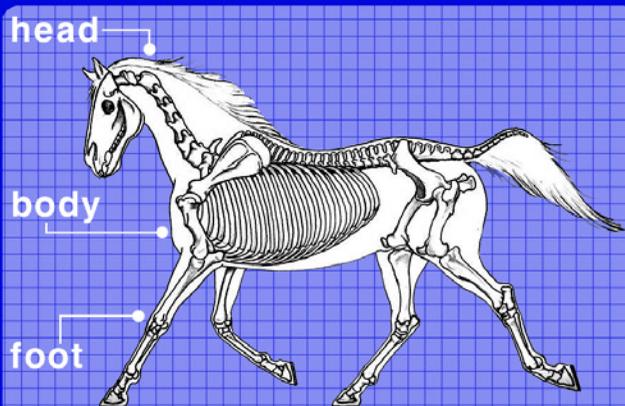
- **СЕЛЕКТОР** – это ссылка на элемент в HTML. Находится в начале CSS-правила и определяет, к каким HTML-элементам применятся свойства и значения из правила.
- **СВОЙСТВО** (Property) – определенная характеристика элемента, которую нужно изменить. Например, размер или цвет.
- **ЗНАЧЕНИЕ** (Value) – цифровое или текстовое обозначение для выбранного свойства. Описывает, как именно свойство будет обработано браузером.



Отличие HTML от CSS

В отличие от **HTML**, который служит для определения структуры и семантики содержимого сайта, **CSS** отвечает за его внешний вид и отображение.

HTML



CSS



ИНСТРУМЕНТЫ ТЕСТИРОВЩИКА

Управление тестированием

Jira – обеспечивает комфортную работу над проектами, отслеживание задач, совместную корректировку документов, планирование и наглядное проектирование.

Redmine – позволяет контролировать проекты, отслеживать задачи, составлять контрольные списки дел, строить диаграммы и графики. Есть бесплатная пробная версия.

TestRail – решение, созданное специально для команд QA. Оно позволяет управлять тестовыми примерами, планами и запусками, а также получать информацию о ходе тестирования в режиме реального времени. С помощью TestRail можно настроить интеграцию с трекерами проблем.

Скриншоты и работа с ними

Snagit – содержит все лучшее из возможного. Приложение вырезает скриншоты, записывает видео и аудио, работает с веб-камерой, создает gif, рисует, пересылает. Есть бесплатная пробная версия.

Recordit API – создает видео и отправляет сразу в приложение, а еще одним кликом превращает его в gif.

Monosnap – редактирует скриншоты, выделяет детали, скрывает конфиденциальную информацию.

GreenShot – свободное ПО для создания скриншотов на Windows. Легкий, гибкий и надежный инструмент.

Генераторы данных

Mockaroo.com (mockaroo.com) Генерирует тестовые данные человека (имя, номер карты и прочее), но и генерирует SQL-запрос.

Bugmagnet – плагин для Chrome и Firefox. Генерирует различные данные для заполнения полей.

Generatedata.com (generatedata.com) Генерирует данные пользователя и формирует запросы.

Генератор изображений (placeimg.com)

Генераторы временных почтовых ящиков (temp-mail.org/ru)

Генератор личности (fakenamegenerator.com/advanced.php)

Генераторы текста и строк (online-generators.ru/text)

Визуальное отображение и интеллект-карты

Coggle (coggle.it) – бесплатное онлайн-приложение с простым интерфейсом и большим набором функций.

Xmind (xmind.net) – кросс-платформенное приложение, доступное в бесплатной и в расширенной платной версиях. Большой его плюс – возможность работать с диаграммами Ганта.

Freemind (sourceforge.net/projects/freemind) – бесплатное приложение для создания диаграмм связей с почти полным набором функций для работы.

Чек-листы

Testpad (ontestpad.com) – инструмент для составления плана тестирования и контроля с помощью списков. Комфортный и гибкий в работе.

Sitechco (sitechco.ru) – онлайн-сервис для ведения чек-листов, позволяющий хранить результаты, просматривать отчеты и статистику. Большое достоинство Sitechco – возможность интеграции с Jira Cloud.

Teamsuccess (teamsuccess.io) – сервис, в котором перечислены некоторые проверки, которые пригодятся при тестировании и помогут структурировать идею.

Google-таблицы очень полезный и эффективный инструмент в умелых руках.

Тестирование API

SoapUI – это консольный тестировщик, с помощью которого легко протестировать API REST и SOAP, а также Web-сервисы.

Postman – отличный инструмент для тестирования API, может работать как расширение Google Chrome.

Katalon Studio – это бесплатный инструмент автоматизированного тестирования UI, API. Katalon поддерживает запросы SOAP и RESTful с различными типами команд.

ПРИМЕРЫ ТЕХНИЧЕСКИХ ВОПРОСОВ НА СОБЕСЕДОВАНИИ

*Вопросы, ответы на которые есть в Шпаргалке
Вопросы кликабельны (не только номера страниц)*

- Что такое тестирование? [Стр. 5](#)
- Цель тестирования? [Стр. 5](#)
- Какие принципы тестирования вы знаете? [Стр. 5](#)
- Когда стоит начинать тестирование? [Стр. 5](#)
- Что такое баг? [Стр. 7](#)
- Опишите жизненный цикл бага. [Стр. 7](#)
- Опишите жизненный цикл ПО. [Стр. 9](#)
- Какие вы знаете методы управления проектами?
[Стр. 14](#)
- Что такое Waterfall? [Стр. 10](#)
- Что такое Scrum, Agile [Стр. 14](#)
- Опишите этапы тестирования? [Стр. 17](#)
- Какие виды тестирования вы знаете? [Стр. 18](#)
- Чем валидация отличается от верификации. [Стр. 19](#)
- Какие бывают требования? [Стр. 20](#)
- Методы тестирования (Черный, белый, серый ящик).
[Стр. 21](#)
- Позитивное и негативное тестирование. [Стр. 26](#)
- Чем фронтенд отличается от бекенда? [Стр. 26](#)
- Функциональное и нефункциональное тестирование. [Стр. 27](#)
- Чем отличается смоук от регрессионного тестирования?
[Стр. 28](#)
- Что такое санитарное тестирование? [Стр. 28](#)
- Какие вы знаете виды тестовой документации? [Стр. 31](#)
- Что такое чек-лист? [Стр. 31](#)
- Что такое use-case? [Стр. 31](#)
- Что такое тест-кейс? [Стр. 32](#)
- Чем чек-лист отличается от тест-кейса. [Стр. 32](#)
- Что такое тест-план? [Стр. 34](#)
- Обязательные и необязательные атрибуты баг-репорта.
[Стр. 36](#)

- Что такое *severity* и *priority*, чем отличаются? [Стр. 36](#)
- Приведите пример бага с низкой серьезностью, но высоким приоритетом. [Стр. 36](#)
- Какие вы знаете техники тест-дизайна? [Стр. 37](#)
- Классы эквивалентности и граничные значения. [Стр. 37](#)
- Что такое таблица принятия решений? [Стр. 38](#)
- Что такое *ad-hoc* тестирование? [Стр. 40](#)
- Что такое исследовательное тестирование? [Стр. 40](#)
- Особенности тестирования мобильных приложений.
[Стр. 41](#)
- Нативные, гибридные, веб приложения. [Стр. 41](#)
- Что такое кросс-браузерное тестирование? [Стр. 43](#)
- Что нельзя протестировать на эмуляторе, а только на реальном устройстве? [Стр. 44](#)
- Какие бывают прерывания? [Стр. 45](#)
- Что можно протестировать через DevTools? [Стр. 45](#)
- Что такое сниффер и зачем он нужен? [Стр. 45](#)
- Что такое кэш и куки? Чем отличаются? [Стр. 45](#)
- Зачем нужно чистить кэш перед тестированием веб-приложений? [Стр. 45](#)
- Что такое SQL? [Стр. 46](#)
- Что вы знаете о NoSQL? [Стр. 46](#)
- Основные команды SQL? [Стр. 48](#)
- Чем HTTP отличается от HTTPS? [Стр. 49](#)
- Опишите, что такое клиент-серверная архитектура?
[Стр. 50](#)
- Из чего состоит HTTP-запрос? [Стр. 50](#)
- Какие вы знаете HTTP-запросы? [Стр. 50](#)
- Чем GET отличается от POST? [Стр. 50](#)
- Что такое REST API? [Стр. 52](#)
- Отличия от SOAP? [Стр. 52](#)
- Что такое JSON/XML? [Стр. 52](#)
- Что такое HTML? [Стр. 53](#)
- Что такое CSS? [Стр. 55](#)

ПРИМЕРЫ **ЛОГИЧЕСКИХ** ВОПРОСОВ НА СОБЕСЕДОВАНИИ

Вопросы для самостоятельной подготовки

- *Почему вы решили стать тестировщиком?*
- *У вас открывается веб-сайт. Белый экран. Больше ничего. Как будете тестировать?*
- *Как будете тестировать кнопку?*
- *Как будете тестировать поле?*
- *Протестируйте маску.
5 позитивных и 5 негативных проверок.*
- *Накидайте проверок на робот-пылесос.
Позитивных и негативных.*
- *Вы тестировщик. К вам приходит заказчик с мобильным приложением. Сколько времени и какие доступы вы попросите на тестирование?*
- *Ваша команда разрабатывает веб-приложение для бронирования путешествий. Оно будет готово через 2 месяца. Когда вам следует начать тестирование?*
- *Вы обнаружили ошибку при тестировании мобильного приложения и сообщили о ней. Разработчик не может её воспроизвести. Что вы можете упустить в этом случае?*
- *Как QA-инженер вы читаете требования для создания тестовых случаев и обнаружили среди них некоторую двусмысленность. Что вы можете сделать в таком случае?*
- *Для чего нужны «правильные» отношения с другими членами команды?*
- *Что такое этика, и почему она должна иметь значение, особенно для QA-инженеров?*
- *Когда QA-инженеру следует обращаться за помощью?*
- *Объясните 7-летнему ребенку, что такое база данных.*