1. Баг — (англ. bug — жук, насекомое) — ошибка (сбой) в программе. Означает «недоработку в компьютерной программе, приводящую к нежелательным или неожидаемым действиям, или же вообще не позволяющую её запустить».
2. Пользовательский сценарий (User Scenario) — это **схема, которая позволяет определить, почему покупатели оказываются на сайте и как реализуют свои планы с помощью вашего продукта**. Чаще всего такие сценарии создаются в формате коротких рассказов о некоем эталонном пользователе, цель которого — удовлетворить свою потребность посредством вашего сайта или приложения.
3. Скилл (skill) — это рабочий, игровой или социальный навык.
4. **Сеть** — это совокупность устройств и систем, которые подключены друг к другу (логически или физически) и общаются между собой.
5. **Локальная** сеть есть даже у вас дома. Все устройства, которые подключены к роутеру (телефоны, планшеты, компьютер), образуют вместе локальную сеть.
6. Пример **глобальной** сети вы можете наблюдать, отправляя поисковый запрос, например, в Яндексе. Подключение к глобальной сети осуществляется через интернет-провайдера.
7. **Устройства** и **среда** — это физические элементы или оборудование сети, которые часто являются видимой частью сети: ноутбуки, ПК, коммутатор, маршрутизатор, кабели для соединения устройств. Те устройства, которые образуют интерфейс между пользователем и коммуникационной сетью, которая предоставляет связь, называются **оконечными устройствами** или **узлами**. К ним относятся ноутбуки, ПК, сетевые принтеры, терминальное оборудование, смартфоны и так далее. Другие устройства, которые служат для соединения узлов, называются **промежуточными устройствами**, к ним относят, например, коммутаторы и маршрутизаторы.
8. **Коммутатор** — это устройство доступа к сети, которое объединяет несколько компьютеров в локальную сеть для обмена данными между ними.
9. **Маршрутизатор** — устройство сетевого взаимодействия, которое осуществляет поиск оптимального пути передачи информации от одного устройства сети до другого. Например, от сервера какого-либо сайта до вашего персонального компьютера.
10. **Серверы** — это узлы, на которых установлено ПО, позволяющее им *предоставлять* другим сетевым узлам какую-либо информацию. Например, доступ к электронной почте, веб-страницам, о чём мы и говорили в примерах.
11. **Клиенты** — это компьютерные узлы с установленным ПО, которое позволяет *запрашивать* и *отображать* полученную с сервера информацию. Например, веб-браузер или приложение электронной почты.
12. Модель **Open Systems Interconnection** (*OSI*) — это скелет, фундамент и база всех сетевых сущностей. Модель определяет сетевые протоколы, распределяя их на 7 логических уровней.Физический — самый низкий — определяет процесс прохождения сигналов через среду передачи между сетевыми устройствами (узлами сети).
13. Канальный — уровень передачи данных.
14. Сетевой — отвечает за передачу данных и управляет маршрутизацией сообщений — передача через несколько каналов связи по одной или нескольким сетям, что обычно требует включения в пакет сетевого адреса.
15. Транспортный уровень — управляет сквозной передачей сообщений между оконечными узлами сети, обеспечивая надёжность и экономическую эффективность передачи данных независимо от пользователя.
16. Сеансовый — обеспечивает обслуживание двух «связанных» на уровне представления данных объектов сети и управляет ведением диалога между ними путём синхронизации, заключающейся в установке служебных меток внутри длинных сообщений.
17. Уровень представления — обеспечивает совокупность служебных операций, которые можно выбрать на прикладном уровне для интерпретации передаваемых и получаемых данных.
18. Прикладной — обеспечивает непосредственную поддержку прикладных процессов и программ конечного пользователя, а также управление взаимодействием этих программ с различными объектами сети.
19. **Интерфейс** — это соглашение о взаимодействии между уровнями одной системы, определяющее структуру данных и способ (алгоритм) обмена данными между соседними уровнями *OSI*-модели.
20. **Протокол** — это совокупность правил, регламентирующих формат и процедуры взаимодействия процессов одноимённых уровней на основе обмена сообщениями.
21. **FTP** (англ. *File Transfer Protocol*) — протокол передачи файлов. *FTP* даёт возможность абоненту обмениваться двоичными и текстовыми файлами с любым компьютером сети. Установив связь с удалённым компьютером, пользователь может скопировать файл с удалённого компьютера на свой или скопировать файл со своего компьютера на удалённый.Например, при работе с файлами сайта для быстрого обмена между ПК и директориями сайта.
22. **HTTP** (англ. *Hypertext Transfer Protocol*) — протокол работы с гипертекстовыми документами.Протокол *HTTP* используется при пересылке *Web*-страниц между компьютерами, подключёнными к одной сети. Также сейчас имеется расширение протокола *HTTP* — **HTTPS** (безопасный протокол передачи гипертекста), который поддерживает шифрование посредством криптографических протоколов *SSL* и *TLS*. *HTTPS*-протокол необходимо использовать, например, на веб-сайтах, где вводится и передаётся конфиденциальная информация (личные данные, детали доступа, реквизиты платёжных карт), на любых веб-сайтах, где используется авторизация, взаимодействие с платёжными системами, почтовыми сервисами. Использование этого протокола позволит предотвратить получение и использование данных третьими лицами.
23. **TELNET**— это протокол удалённого доступа.  *TELNET* даёт возможность абоненту работать на любой ЭВМ, находящейся с ним в одной сети как на своей собственной: запускать программы, менять режим работы и так далее.
24. **TCP** (*Transmission Control Protocol*) — протокол контроля передачи информации.Когда осуществляется передача от компьютера к компьютеру через Интернет, *TCP* работает на верхнем уровне между двумя конечными системами, например, браузером и веб-сервером. *TCP* осуществляет надёжную передачу потока байтов от одного процесса к другому. *TCP* реализует управление потоком, управление перегрузкой, рукопожатие, надёжную передачу.
25. **Ethernet**— протокол определяет формат кадров и управление доступом к среде — на канальном уровне модели *OSI*.
26. *Web***-технологии** — комплекс технических, коммуникационных, программных методов решения задач организации совместной деятельности пользователей с применением сети Интернет.
27. **Веб-страницы** — это файлы в формате «неформатированный текст» (*plain text*, текст в *ASCII*-кодах), распознаваемые любой операционной системой.
28. **Веб-сайт** — это совокупность логически связанных между собой веб-страниц, или просто одна страница. Доступ в Интернет осуществляется с помощью специальной программы — **браузера**.
29. Статическими — назначается конкретному устройству и не изменяется. Статические IP-адреса обычно имеют все веб-сайты. Постоянный адрес служит гарантией того, что пользователь получит доступ к тому же серверу, что и ранее. Динамическими — адрес, который меняется при каждом подключении к сети. Обычно «выдаётся» пользователям. Чаще всего набор IP-адресов у провайдера ограничен, поэтому, когда новое устройство подключается к сети, ему выдаётся любой свободный адрес. После его отключения этот адрес может быть присвоен другому устройству. Стоит отметить, что такие адреса считаются более безопасными по сравнению со статическими: затрудняется отслеживание компьютера и других устройств, подключенных к сети.
30. **DNS**— это огромная таблица, в которой хранятся имена сайтов и соответствующие им *IP*. Когда вы вводите адрес сайта, браузер обращается к *DNS*-серверу, просматривает таблицу данных, находит совпадение и получает *IP*. Таким образом, *DNS* — это **технология, которая помогает браузеру найти правильный сайт по доменному имени**.
31. *IP***-адрес**— это сокращение от фразы *Internet Protocol Address*, что означает — уникальный сетевой адрес.
32. [*Bubble*](https://bubble.io/) служит для создания веб-приложений без использования кода. Инструмент воспроизводит все основные опции веб-программирования в понятном визуальном интерфейсе.
33. [*Tilda*](https://tilda.cc/ru/) позволяет создавать впечатляющие, красивые и легкие в управлении сайты, интернет-магазины, лендинги и спецпроекты без использования программирования.
34. [*WordPress*](https://ru.wordpress.org/) — это одна из наиболее популярных CMS-систем управления контентом на сайте (создание и публикация записей, размещение виджетов, изменение дизайна, расположение и отображение различных элементов и так далее).
35. ***HTML***(*HyperText Markup Language* — язык гипертекстовой разметки) — это не язык программирования!
36. В первую очередь это логическая разметка страницы: описывает структуру страницы, а не её поведение.
37. ***CSS*** (Cascading Style Sheets — каскадные таблицы стилей) — язык таблиц стилей, который позволяет прикреплять стиль, например, шрифты и цвет.
38. ***JavaScript*** или ***JS —***Логический язык программирования, который можно использовать для изменения содержимого веб-сайта, заставить сайт вести себя по-разному **в ответ на действия пользователя**. Общее использование — это окна подтверждения, призывы к действию и добавление новых идентификаторов к существующей информации.
39. "***Сервер*** (ударение на первую «е») — это компьютер, который обрабатывает запросы, поступающие от других компьютеров («клиентов») в рамках какого-либо сетевого протокола. Английский глагол ***to serve***, от которого произошло слово «сервер», означает попросту «обслуживать».

**Тег** — это базовый элемент языка разметки, **основа** HTML-документа. **Тело сайта** содержит все видимые элементы. Обычно содержимое сайта размещают внутри блоков, определяемых тегами <div> </div>.

Для заголовков используются теги <h1> </h1>, <h2> </h2> и так далее до <h6> </h6> (чем меньше цифра, тем «важнее» заголовок).

Обычный текст должен включаться в теги <p> </p> — это теги для выделения параграфов.

Самый важный тег, ради которого всё и придумывалось — <a href="ссылка">текст ссылки</a> — поддерживает возможность открытия документа, на который указывает значение атрибута href.

<form> </form> — форма, которую заполняет пользователь.

<button> </button> — кнопка.

<b> </b> и <strong> </strong>— отображает текст полужирным.

<i> </i> и <em></em> — отображает текст курсивом.

<table> </table> — таблица. Внутри таблица состоит из тегов строк <tr> </tr> и тегов ячеек <td>содержимое</td>.

<ul> </ul> и <ol> </ol> — список со значками (маркированный) и список с цифрами (нумерованный). Отдельный элемент списка размещается в тегах <li>элемент списка </li>.

Из **не**парных тегов часто встречается только два:

<br> — тег разрыва строки.

<img src="https://link.2pic/pic.png"> — тег, который указывает браузеру загрузить и отразить изображение по ссылке в атрибуте src.

Отсутствие вредных привычек, энергичность, инициативность, самостоятельность, ответственность, коммуникабельность, быстрая обучаемость

***Жизненный цикл программного обеспечения (Software Life Cycle Model)*** — это период времени, который начинается с момента принятия решения о создании программного продукта (средства) и заканчивается в момент его полного изъятия из эксплуатации.

**программный продукт** (software product): Совокупность компьютерных программ, процедур и, возможно, связанных с ними документации и данных

* **Каскадная** (водопад) — модель, представляющая собой последовательное выполнение каждого из этапов жизненного цикла.
* **V-модель**— разработка через тестирование, тесты внедряются в каждый из этапов производства.
* **Спиральная модель** — предполагает цикличное повторение каждой стадии производства из раза в раз, вплоть до выпуска продукта.
* **Agile**— гибкий подход к процессам разработки и производства, в основе которого лежат быстрый результат и взаимодействие с людьми.
* **Scrum** — гибкая методика управлениями всеми стадиями жизненного цикла.

**техники тест-дизайна** — это совокупность правил, которые позволяют правильно определить список проверок для тестирования

***Манифест гибкой разработки программного обеспечения (Agile Manifesto)*** — основной документ, содержащий описание ценностей и принципов гибкой разработки программного обеспечения.

**Чёрный ящик**

* Не подразумевает понимание внутреннего устройства системы.
* Позволяет быстро выявить ошибки в функциональных спецификациях.
* Тестирование проходит «с позиции» пользователя.
* Не требует от тестировщика дополнительных навыков.
* Составлять тест-кейсы можно сразу после подготовки спецификации.

**Пример:** водитель умеет только управлять машиной.

### ****Белый ящик****

* Оптимизация кода путём нахождения скрытых ошибок.
* Более тщательное тестирование, потому что тестами обычно покрыт весь код.
* Возможность автоматизации тест-кейсов.

**Пример:** водитель не только умеет управлять машиной, но и чинить ее самостоятельно.

### ****Серый ящик****

* Включает в себя плюсы тестирования черного и белого ящиков.
* Возможность проектировать и использовать более сложные сценарии тестирования.
* Совместная работа тестировщика и разработчика, что позволяет на начальном этапе убрать избыточные тест-кейсы. Это сокращает время функционального и нефункционального тестирования и положительно влияет на общее качество продукта.

**Пример:** водитель может управлять машиной и исправлять мелкие поломки (например, менять колесо при необходимости).

**Модульное тестирование** (также — компонентное тестирование) — проверка функциональности отдельных модулей (модулей программ, объектов, классов, функций и т. д.) и поиск дефектов в отдельных частях приложения.

Давайте рассмотрим небольшой пример.Представьте, что вы — инженер-строитель и должны проверить, правильно ли работают окна, которые планируется установить в здании. Что вы проверите на начальном этапе тестирования, до того как окна будут установлены? Убедитесь, что работают ручки и механизмы, переводящие створки в разные положения. В данном случае ручки выступают в роли компонентов — наименьших элементов, которые можно протестировать отдельно.

**Интеграционное тестирование** — проверка связи между компонентами и взаимодействия между разными системами (например, смежной системой).

Используется для выявления дефектов при взаимодействии отдельных компонентов системы, объединённых в крупные блоки.

Внутри интеграционного тестирования есть два уровня:

|  |  |
| --- | --- |
| ****Компонентный интеграционный уровень**** Осуществляется проверка взаимодействия между компонентами системы после проведения компонентного тестирования. ****Системный интеграционный уровень**** Осуществляется проверка взаимодействия между разными системами после проведения системного тестирования. |  |

Важно также сказать, какие существуют подходы к интеграционному тестированию.

* **Подход «Снизу вверх»**: все низкоуровневые модули собираются воедино и затем тестируются, после чего собирается следующий уровень модулей.
* **Подход «Сверху вниз»**: первоначально тестируются все высокоуровневые модули, затем постепенно добавляются низкоуровневые.
* **Подход «Большой взрыв»**: все разработанные модули собираются вместе в виде законченной системы, а затем проводится интеграционное тестирование.
* Вернёмся к примеру с инженером-строителем.
* Он во всём убедился и уверен, что окна правильно собраны, все компоненты функционируют как надо. Пришло время проверить, хорошо ли они работают с другими частями здания и друг с другом. Например, по нажатию кнопки пульта одновременно все окна должны закрываться или переводиться в положение «проветривание».
* **Системное тестирование** — тестирование как функциональных, так и нефункциональных требований и проверка на наличие дефектов во всей системе.
* Можно выделить два подхода к системному тестированию.

### ****На базе требований****

Для каждого требования пишутся тестовые случаи, которые проверяют его выполнение.

### ****На базе случаев использования****

На основе представления о способах использования продукта создаются случаи использования системы (по конкретному случаю можно определить один или несколько сценариев). На проверку каждого сценария пишутся тест-кейсы, которые должны быть протестированы.

**Приёмочное тестирование** — формальный процесс тестирования, в ходе которого проводится определение соответствия системы приёмочным критериям, а также принимается решение о принятии работ и приложения в целом.

Инженером выполнен ряд тестов по техническим спецификациям, найденные дефекты устранены. Пора продемонстрировать работу заказчику. Для этого используется **приемочное тестирование**: оно проводится на основе набора типичных тестовых сценариев, разработанных в соответствии с требованиями к данному приложению.

**Цель приёмочного тестирования** — определение готовности продукта.

Этот тип тестирования выполняется в той среде, где будет постоянно работать объект.

Виды приемочного тестирования:

→ **Пользовательское приёмочное тестирование.**

→ **Альфа-тестирование.** Проводится разработчиками или командой тестировщиков на поздней стадии разработки с имитацией реального использования продукта.

→ **Бета-тестирование.** Проводится небольшой группой авторизированных пользователей с целью собрать первые фидбеки.

Многие из вас, возможно, сталкивались с тем, что для игры сначала выпускалась бета-версия. И если вы ею пользовались, поздравляем — вы участвовали в приёмочном тестировании.

Все виды тестирования можно разделить на три группы:

|  |  |
| --- | --- |
| **1** | **Функциональные**  Основываются на функциях, которые выполняются системой, и могут проводиться на всех уровнях тестирования. |
| **2** | **Нефункциональные**  Основываются на проверке характеристик системы, которые могут быть изменены различными величинами. |
| **3** | **Связанные с изменениями**  Основываются на проверке исправлений багов (дефектов). |

**ISTQB** (***I***nternational ***S***oftware ***T***esting ***Q***ualifications ***B***oard) — международная организация, которая занимается сертификацией специалистов в области QA.

Выполнение функционального тестирования — строгий процесс, который состоит из следующих этапов.

|  |  |
| --- | --- |
| **1** | ****Подготовка**** Прежде всего, на подготовительном этапе проводится:   * анализ всех документов о том, как должна выполняться система (например, функциональных и бизнес-требований, технических спецификаций); * разработка и утверждение плана тестирования и  сценариев тестирования; * согласование сроков проекта, количества спринтов (если мы следуем методологии Agile); * оценка возможных бизнес-рисков (проекта, продукта и процесса).   В идеале задачи на этом этапе выполняются совместно с представителями заказчика. |
| **2** | ****Выполнение**** Согласно подготовленным сценариям испытаний и исходным данным, проводится фактическое тестирование. Все обнаруженные ошибки регистрируются в системе отслеживания ошибок, например JIRA. |
| **3** | ****Составление отчёта**** Когда команда тестирования завершила выполнение тестовых сценариев, готовится протокол (отчёт). Наряду с данными и их анализом он также может содержать, например, список рекомендаций по улучшению системы. |

Существуют различные **виды тестирования,** которые могут в себя включать функциональные тест-кейсы. К основным относятся следующие:

* **Регрессионное тестирование** — это тестирование функциональности продукта после исправления ошибок или реализации новой функциональности.
* **Интеграционное тестирование.** Позволяет проверить взаимодействие между компонентами системы.
* **Дымовое тестирование.** С помощью коротких циклов тестов выявляет правильность работы основных функций системы.
* **Системное тестирование.** Выполняется на полной, интегрированной системе с целью проверки соответствия всей системы исходным требованиям.
* **Тестирование документации.** Связано с проверкой документов, например, на соответствие принятым стандартам.
* **Тестирование мобильных приложений.** Ещё одно направление в обеспечении качества систем, которое ориентировано на программы, предназначенные для той или иной платформы (*Android*, *iOS* и др.).
* **Тестирование установки.** Выполняется для проверки процесса инсталляции/деинсталляции программ.

**Нефункциональное тестирование** — тестирование, проводимое для оценки того, что компонент или система соответствует нефункциональным требованиям.

К основным критериям, характеризующим продукт, обычно относят следующее:

* **Безопасность** (защищённость пользовательских данных, целостность приложения/системы).
* **Надёжность** (реакция системы на непредвиденные ситуации).
* **Удобство** (удобство работы с приложением с точки зрения пользователя).
* **Масштабируемость** (требования к горизонтальному или вертикальному масштабированию приложения).
* **Производительность** (работоспособность системы под разными нагрузками).
* **Портируемость/Кроссплатформенность** (переносимость приложения на различные платформы).

### ****Тестирование безопасности****

Проверка стратегии тестирования, которая используется для проверки безопасности системы, а также для анализа рисков, которые могут быть связаны с обеспечением целостности подхода к защите приложения (системы) от атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

### ****Тестирование производительности****

Проверка работы приложения системы под нагрузкой.

### ****Тестирование установки****

Проверка успешности установки приложения, его настройки и удаления. Снижает риски потери пользовательских данных, потери работоспособности приложения и т. д.

### ****Тестирование удобства использования****

Характеризует систему с точки зрения удобства использования конечным пользователем.

### ****Конфигурационное тестирование****

Исследование работоспособности программной системы в условиях различных программных конфигураций (поддерживаемых драйверах, различных конфигурациях компьютеров и т. д.).

### ****Тестирование на отказ и восстановление****

Исследование программной системы на предмет восстановления после ошибок, сбоев. Оценивание реакции защитных свойств приложения.

Какие виды тестирования связанные с изменениями вы знаете?

* **Дымовое** **тестирование**, или смоук-тестирование.
* **Регрессионное тестирование**.
* **Тестирование сборки**.
* **Санитарное тестирование**, или проверка согласованности (исправности).

**Регрессионное тестирование** (от англ. regression testing) — это тип тестирования, связанный с изменениями программного обеспечения, позволяющий определить, что изменения не повлияли на работоспособность ранее протестированных участков исходного кода.

### ****Перепроверить всё****

Как следует из названия, тест-кейсы из тест-комплектов выполняются повторно, чтобы гарантировать работу программного кода «как раньше». Это дорогостоящий метод, поскольку он требует больше времени и ресурсов по сравнению с другими методами.

### ****Выбрать сценарий регрессионного тестирования****

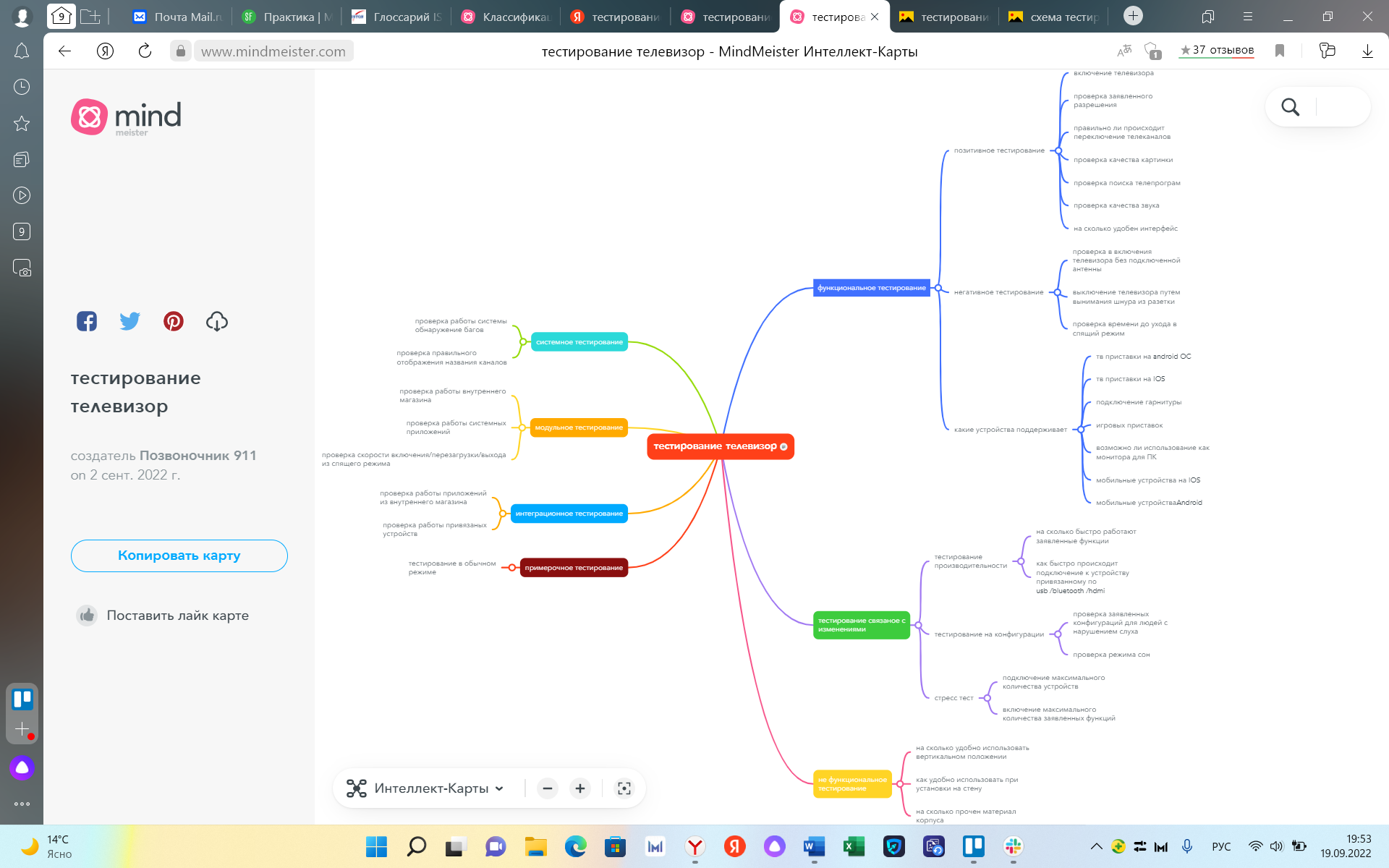
В этом методе тест-кейсы выбираются из тест-комплектов для повторного выполнения. Не весь набор перезапускается. Выбор тестовых случаев осуществляется на основе изменения кода в модуле.

### ****Приоритизировать тестовый сценарий****

Тестовые случаи с высоким приоритетом выполняются в первую очередь. Приоритет тестового случая зависит от его критичности и влияния на продукт, а также от наиболее часто используемой функциональности продукта.

### ****Комбинировать техники****

Гибридная техника представляет собой комбинацию второй и третьей техник. Вместо того чтобы выбирать весь набор тестов, выбирайте только те тестовые случаи, которые выполняются повторно в зависимости от их приоритета.



**Верификация** — это проверка, соответствует ли продукт спецификации.

**Валидация** — это проверка, соответствует ли продукт требованиям и ожиданиям.

* **Тестирование** — это процесс, состоящий из всех действий жизненного цикла, как статических, так и динамических, связанных с планированием, подготовкой и оценкой программных продуктов и связанных с ними рабочих продуктов, а также определением того, что они удовлетворяют заданным требованиям, для демонстрации их соответствия назначению и обнаружения дефектов. Тестирование — это процесс, а не отдельное действие, в него входит разные виды деятельности. (*ISTQB*).
* **Тестирование** — это не действие, а интеллектуальная дисциплина, имеющая целью получение надёжного программного обеспечения, без лишних усилий на его проверку. (Борис Бейзер «Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем»).

Вот несколько определений того, что такое тест-дизайн из *ISTQB*:

* + **Тест-дизайн** — это процесс преобразования общих целей тестирования в осязаемые условия тестирования и тест-кейсы.
  + **Тест-дизайн** — это документ, определяющий условия тестирования (элементы покрытия) для элемента тестирования, подробный подход к тестированию и определение связанных с ним тест-кейсов.
  + **Тест-дизайн** — это деятельность, причём активная, направленная на определение тестовых случаев (тест-кейсов) исходя из целей, условий тестирования и определенных критериев качества.
* **Верификация** (*verification*) — подтверждение экспертизой и предоставлением объективных доказательств того, что заданные требования выполнены.
* **Валидация** (*validation*) — подтверждение экспертизой и предоставлением объективных доказательств того, что заданные требования для конкретного предполагаемого использования или применения были выполнены.
* **Тест дизайн** (*test design*) — это процесс преобразования общих целей тестирования в осязаемые условия тестирования и тест-кейсы.
* **Тест анализ** (*test analysis)* — это процесс изучения того, что может быть использовано для получения тестовой информации. Эта основа для тестов называется тестовый базис (тестовая база).
* **Тестовый базис** (*test basis*) — тестовый базис включает всё, на чём основаны тесты. Это может быть системное требование, техническое задание, сам код (для структурного тестирование) или бизнес-процесс. Иногда тесты могут быть основаны на опытном знании пользователем системы, которое может быть не задокументировано.
* **Объект тестирования** (*test object* или *test item*) — компонент или система, которую мы будем тестировать. Простыми словами — это то, что будем тестировать.

**Тест-кейс** — это набор входных значений, предусловий выполнения, ожидаемых результатов и постусловий выполнения, разработанный для определённой цели или тестового условия, таких как выполнения определённого пути программы или же для проверки соответствия определённому требованию.Другими словами, тест-кейс — это документ, который описывает условия, шаги и ожидаемый результат, благодаря которым тестировщик проверяет, работает ли продукт в соответствии со спецификацией.

**Позитивный тест-кейс** проверяет, что при использовании корректных данных, приложение правильно выполнило функцию в соответствии со спецификацией.  
Пример: при вводе правильного формата электронной почты *student@skillfactory.ru* при регистрации поле «электронная почта» подсвечено зелёным цветом.

**Негативный тест-кейс** проверяет, что при использовании как корректных, так и некорректных данных, вызываемая функция не выполняется и срабатывает валидация.  
Пример: при вводе неправильного формата электронной почты *student@skillfactory* при регистрации на портале всплывает сообщение: «Пожалуйста, введите электронную почту в формате [*student@skillfactory.ru*»](mailto:student@skillfactory.ru).

1. **Тест Кейс №/ID** — это уникальный идентификационный номер тест-кейса.
2. **Проект** — название вашего проекта\приложения или ссылка на него.
3. **Компонент** — компонент или часть системы, к которому относится этот тест-кейс.
4. **Дата** — дата написания тест-кейса.
5. **Тестировщик** — имя и фамилия тестировщика.
6. **Проверил (опционально)** — имя и фамилия *QA*-менеджера, который проверил тест-кейс.
7. **Дата проверки (опционально)** — дата проверки тест-кейса *QA*-менеджером.
8. **Описание** — описание тест-кейса, что конкретно мы проверяем данным тест-кейсом.
9. **Предусловия** — необходимые начальные условия (аккаунт пользователя создан, домашняя страница приложения открыта в браузере и так далее).
10. **Тестовые данные** — ваши тестовые данные (пароль, логин, имя, фамилия, номер счета, город и так далее).
11. **Шаг** — номер шага по порядку от 1 (1, 2, 3 и так далее).
12. **Описание шага** — описание действий, которые мы выполняем.
13. **Ожидаемый результат** — ожидаемое состояние системы, исходя из спецификации (откроется страница, будет выведено сообщение об ошибке на экран и так далее). Для данного шаблона ожидаемый результат последнего шага совмещён **с постусловием**.

**Система управления тестовой документацией** (Test Management Software) — это специальная программа для создания и выполнения тест-кейсов, создания тест-планов и планирования тестовой деятельности, а также создания и хранения отчётов о тестировании.

Основная часть тест-кейса в *Zephyr Squad* находится в секции *Test Details*:

* + *Test Step* — тестовый шаг (описание шага/действий),
  + *Test Data* — тест-данные (данные, которые мы используем в тест-кейсе),
  + *Test Results* — результаты теста (ожидаемый результат и/или постусловие).
* **Тест-кейс** (*Test Case*) — это набор входных значений, предусловий выполнения, ожидаемых результатов и постусловий выполнения, разработанный для определённой цели или тестового условия, таких как выполнение определённого пути программы или же для проверки соответствия определенному требованию. Другие названия: тестовый случай, тестовая ситуация, вариант тестирования.
* **Позитивный тест-кейс** проверяет, что при использовании корректных данных приложение правильно выполнило функцию в соответствии со спецификацией.
* **Негативный тест-кейс** проверяет, что при использовании как корректных, так и некорректных данных вызываемая функция не выполняется, и срабатывает валидация.
* **Система управления тестовой документацией** (*Test Management Software*) — это специальная программа для создания и выполнения тест-кейсов, создания тест-планов и планирования тестовой деятельности, а также создания и хранения отчётов о тестировании.
* **Тестовый набор/комплект** (*Test Suite*) — несколько (набор) тест-кейсов, объединённых между собой по типу тестирования, по приоритету, по тестируемой функциональности или модулю.
* **Тест сценарий** — в *ISTQB* также можно найти термины тест сценарий (*test scenario*) и спецификация тестовой процедуры (*test procedure specification*), которые обозначают документ с последовательностью действий для выполнения тестов, также известные как тестовый сценарий (*test script*) или ручной тестовый сценарий (*manual test script*). Некоторые из этих терминов выглядят одинаково при переводе на русский язык, что может создать путаницу.

**Чек-лист**— это список шагов, действий или перечень функциональности, который позволяет тестировщику убедиться в корректной работы приложения.

**Интеллект-карта (mind map)**, известная также как **карта мыслей**, **ассоциативная карта** или **диаграмма связей** — это метод структуризации концепций с использованием графической записи в виде диаграммы.

**Краткие правила составления интеллектуальных карт:**

* Объекты одного уровня, имеющие одного и того же родителя, разделяются по одному признаку. В сумме эти объекты должны составлять единое целое. Например, объекты первого уровня могут быть компонентами системы.
* Объекты второго уровня могут быть функциями компонентов и так далее. Так, раз за разом добавляя новый уровень, можно дойти до полного представления системы на интеллектуальной карте.
* **Интеллект-карта** (*mind map*) известная также как карта мыслей, ассоциативная карта или диаграмма связей — метод структуризации концепций в виде схемы, на которой изображены слова, идеи, задачи или другие понятия, связанные ветвями, отходящими от центрального понятия или идеи.
* **Чек-лист** — это тестовый артефакт, содержащий список проверок. Простыми словами, чек-лист — это список шагов, действий или перечень функциональности, который позволяет тестировщику убедиться в корректной работе приложения.
* **Тестирование на основе чек-листа** — это одна из техник тест-дизайна на основе опыта, когда опытный тестировщик использует лист, который содержит список правил или проверок, на основе которых можно протестировать данный продукт.
* **Веб-приложение** (*web*) — приложение, в котором клиент взаимодействует с веб-сервером при помощи браузера.
* **Десктопное** (*desktop*) приложение — это программа, которая способна работать на компьютере, используя только его аппаратные ресурсы (код самого приложения находится локально на рабочем компьютере).
* **Agile** — обобщающий термин для целого ряда подходов и практик, основанных на ценностях Манифеста гибкой разработки программного обеспечения и 12 принципах, лежащих в его основе. К гибким методологиям, в частности, относят *Scrum*, *Kanban*, экстремальное программирование (*XP*) и другие.
* **RUP** (*Rational Unified Process*) — методология разработки программного обеспечения, созданная компанией *Rational Software*. Была разработана для разработки больших программных систем.
* **Тест-план** (План тестирования) — это документ, описывающий весь объём работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания, конфигурации, знаний и навыков тестировщиков, а также оценки рисков и вариантов их разрешения.
* **Критерии входа** (*Entry Criteria*) и **критерии выхода** (*Exit Criteria*) — критерии, определяющие, когда тестирование должно начинаться и когда оно завершается.
* **Мастер тест план** — высокоуровневый (не детальный) документ, описывающий объём работ по тестированию, ссылки на внутренние и внешние документы, описание требуемых ресурсов и так далее.
* **Детальный тест-план** — детальный документ, который описывает план тестирования на определённом уровне тестирования или для определённого вида тестирования.

**Отчёт по тестированию**— документ, содержащий в себе информацию о выполненных тестах, результатах их выполнения, открытых дефектах и выводах о качестве разрабатываемого приложения. Отчёт по тестированию тесно связан с таким понятием, как фаза тестирования.

**Фазы тестирования** или **жизненный цикл тестирования приложения** (*STLC*) — это определённый набор задач, объединённых в контролируемую фазу проекта. В тестировании можно выделить следующие фазы, которые коррелируются с жизненным циклом продукта:

* + фаза планирования,
  + фаза анализа,
  + фаза проектирования тестов,
  + фаза реализации и выполнения тестов,
  + фаза анализа результатов тестирования (оценка критериев выхода) и отчётов,
  + фаза завершения тестирования

**Тест контроль** — это постоянная активность, потому что нам надо сравнивать фактический прогресс с запланированным прогрессом, а также предоставлять отчёты о текущем состоянии тестирования (включая любые изменения или отклонения от плана) менеджеру проекта и заказчикам. А также принимать меры по корректировке, чтобы двигаться в соответствии с планом.

* **Фазы тестирования** или **жизненный цикл тестирования приложений** (*STLC — Software Test Life Cycle*) *—* это последовательность действий, проводимых в процессе тестирования, с помощью которых гарантируется качество программного обеспечения и его соответствие требованиям. Обычно разбиты на фазы.
* **Базис тестирования** (тестовая база) *—* это различные требования, спецификации, архитектурные диаграммы, спецификации интерфейсов, сценарии использования, отчёты по анализу рисков. Тестовым базисом может быть даже исходный код.

**Разработка тестов методом чёрного ящика** (black box test design technique) — это техника создания и/или выбора тестовых сценариев, основанная на анализе функциональной или нефункциональной спецификации компонента или системы без знания внутренней структуры.

**Функциональное тестирование** — это тестирование программного обеспечения в целях проверки реализуемости функциональных требований.

**Нефункциональное тестирование** — это тестирование программного приложения или системы на предмет их нефункциональных требований: способа работы системы, а не конкретного поведения этой системы. Например, тестирование безопасности, тестирование удобства пользования, нагрузочное тестирование и так далее.

**Базис тестирования** (тестовая база/тест-базис) — это различные требования, спецификации, архитектурные диаграммы, спецификации интерфейсов, сценарии использования, отчёты по анализу рисков. Тестовым базисом может быть даже исходный код.

**Техники чёрного ящика** (*black box*) — техники, основанные на спецификации. Чёрным ящиком он называется потому, что они рассматривают программное обеспечение как чёрный ящик с входом и выходом, но они не знают, как система или компонент структурирован внутри коробки.

**Техники белого ящика** (*white box*) — техники, основанные на структуре (которые также являются динамическими, а не статическими), используют внутреннюю структуру программного обеспечения для создания тест-кейсов. Их называют техниками белого ящика или стеклянного ящика (подразумевая, что вы можете заглянуть внутрь системы), так как они требуют знаний о том, как реализовано программное обеспечение, как оно работает.

**Техники, основанные на опыте** — техники, которые используют знания, навыки и опыт людей (тестировщиков, бизнес аналитиков, пользователей и так далее), являющихся основным фактором, влияющим на условия тестирования и тест-кейсы.

**Попарное тестирование** (*pairwise*) — тест-дизайн методом чёрного ящика, в котором в тест-кейсах выполняют все возможные отдельные комбинации каждой пары входных параметров.

**RUP** (*Rational Unified Process*) — методология разработки программного обеспечения, созданная компанией *Rational Software*. Была разработана для разработки больших программных систем.

**Тестовое покрытие** — степень, выраженная в процентах, до которой указанный элемент покрытия был протестирован тестовым набором.

**Негативное тестирование** направлено, прежде всего, на:

* + проверку устойчивости системы к различным воздействиям;
  + валидацию неверных данных;
  + обработку исключительных ситуаций.

Сценарии **позитивного тестирования**, в свою очередь, направлены на проверку работы системы с теми типами данных, для которых она разрабатывалась.

**таблица причинно-следственных решений (cause-effect decision table)**: См. таблица решений.  
  
**таблица решений (decision table):**Таблица, отражающая комбинации входных данных и/или причин с соответствующими выходными данными и/или действиями (следствиями), которая может быть использована для проектирования тестовых сценариев.

**Таблицы альтернатив** — хороший способ записи сложных бизнес-правил, которые должны быть реализованы в системе. В процессе создания таблицы тестировщик определяет условия (входы) и результирующие действия системы (выходы). Пары условий и действий образуют строки таблицы, при этом условия указываются сверху, а действия — снизу. Каждый столбец представляет собой бизнес-правило с уникальной комбинацией условий и действий, связанных с этим правилом.

**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.

**Предугадывание ошибки** — это ситуация, при которой тестировщик думает над тем, какие ошибки могли быть допущены в процессе разработки, а также определяет пути их появления, используя интуицию, знания и опыт.

***Pairwise Testing*** — техника составления уникальных комбинаций значений для нескольких параметров таким способом, который позволяет составить наименьшее число комбинаций, при этом обязательно выполняется условие о наличии всех возможных парных комбинаций значений всех параметров

* **POST /api/item** — для добавления нового товара. Информация о новом созданном товаре передаётся в теле запроса. В ответ на этот запрос при успешном добавлении нового товара в магазин система должна вернуть в теле ответа информацию о созданном товаре, включая его идентификатор. Например, идентификатор 3050.
* **GET /api/item/3050** — для получения информации о товаре с идентификатором 3050. Информация о товаре возвращается в теле ответа.
* **PUT /api/item/3050** — для изменения информации о товаре с идентификатором 3050. Тело запроса может содержать как только обновленные данные, так и полностью всю информацию, включая обновлённую.
* **DELETE /api/item/3050** — для удаления товара с идентификатором 3050. И, как уже указывалось выше, формат, в котором представлено сообщение в теле ответа или в теле запроса, может быть *JSON*, *XML*, *HTML.*

**SOAP** — это протокол обмена структурированными сообщениями в распределенной среде. [Стандарт](https://www.w3.org/TR/soap12/) определяет формат сообщений, базирующийся на XML. SOAP работает поверх HTTP-протокола, хотя могут использоваться и другие протоколы прикладного уровня, такие как FTP, SMTP.