**Основы компьютерной грамматности**

**Hardware** и **Software** — это два основных компонента любого компьютера или устройства. Они тесно связаны между собой, но выполняют разные функции. Давайте разберем их подробнее:

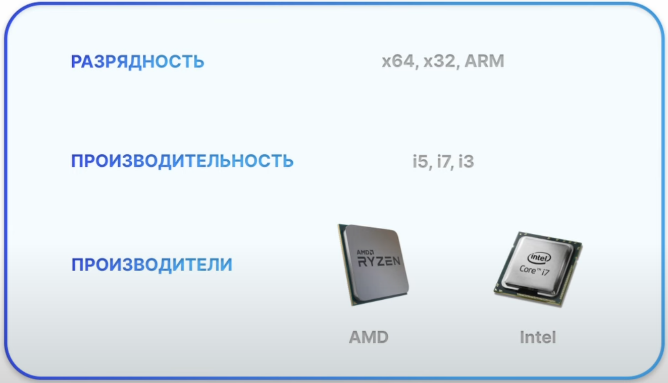


**1. Hardware (Аппаратное обеспечение, "железо")**

Это физические компоненты компьютера или устройства, которые можно потрогать. Hardware обеспечивает выполнение задач, но без программного обеспечения он бесполезен.

**Примеры Hardware:**

1. **Процессор (CPU)**: "Мозг" компьютера, выполняет вычисления.

****

1. **Оперативная память (RAM)**: Временное хранилище данных для быстрого доступа.



1. **Жесткий диск (HDD) или SSD**: Постоянное хранилище данных.

****

1. **Материнская плата(Motherboard)**: Соединяет все компоненты компьютера.

**A black computer motherboard with a box

AI-generated content may be incorrect.**

1. **Видеокарта (GPU)**: Обрабатывает графику.

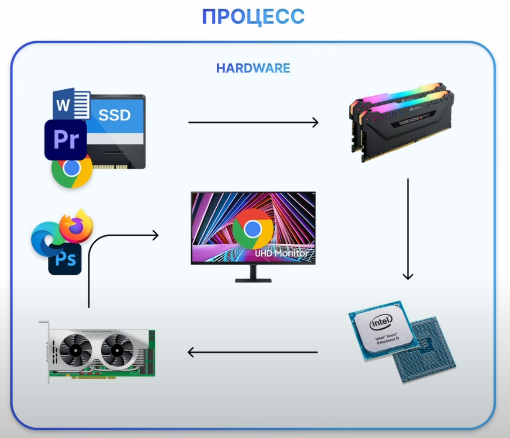
****

1. **Устройства ввода/вывода**: Клавиатура, мышь, монитор, принтер.

A computer monitor and keyboard

AI-generated content may be incorrect.

A computer hardware with different components

AI-generated content may be incorrect.

**Роль Hardware:**

* Обеспечивает физическую основу для работы компьютера.
* Выполняет задачи, которые ему "поручает" программное обеспечение.

**2. Software (Программное обеспечение)**

Это набор программ, которые управляют аппаратным обеспечением и позволяют пользователю выполнять задачи. Software — это то, что "оживляет" железо.

**Примеры Software:**

1. **Операционная система (ОС)**: Windows, macOS, Linux, Android, iOS.
2. **Приложения**: Браузеры (Chrome, Firefox), текстовые редакторы (Word), игры.
3. **Драйверы**: Программы, которые позволяют ОС взаимодействовать с железом.
4. **Утилиты**: Вспомогательные программы (например, антивирусы, архиваторы).

A screenshot of a software

AI-generated content may be incorrect.

**Роль Software:**

* Управляет аппаратным обеспечением.
* Обеспечивает интерфейс для взаимодействия пользователя с компьютером.
* Позволяет выполнять конкретные задачи (работать, учиться, играть).

**Operating System (OS)** — это как "дирижер" вашего компьютера или устройства. Она управляет всеми аппаратными компонентами (железом) и обеспечивает их слаженную работу, чтобы вы могли использовать компьютер для своих задач.

**A diagram of software

AI-generated content may be incorrect.**

**Примеры популярных ОС:**

1. **Windows**:
   * + Разработчик: Microsoft.
     + Основное использование: Персональные компьютеры, ноутбуки, серверы.
     + Особенности: Широкая поддержка программного обеспечения, удобный интерфейс.
2. **Linux**:
   * + Разработчик: Сообщество (разные дистрибутивы: Ubuntu, Fedora, Debian и др.).
     + Основное использование: Серверы, разработка, embedded-системы.
     + Особенности: Открытый исходный код, высокая гибкость и безопасность.
3. **macOS**:
   * + Разработчик: Apple.
     + Основное использование: Компьютеры Mac.
     + Особенности: Интеграция с экосистемой Apple, стабильность, удобство для творческих задач.
4. **Android**:
   * + Разработчик: Google.
     + Основное использование: Смартфоны, планшеты, умные устройства.
     + Особенности: Открытая платформа, поддержка множества приложений.

**Android использует ядро Linux** как основу для управления аппаратными ресурсами (процессор, память, устройства ввода/вывода).

**Преимущества Linux:**

* + - Высокая стабильность и безопасность.
    - Поддержка множества устройств и архитектур.
    - Открытый исходный код, что соответствует философии Android.

1. **iOS**:
   * + Разработчик: Apple.
     + Основное использование: iPhone, iPad.
     + Особенности: Закрытая экосистема, высокая производительность и безопасность.

**iOS** используется ядро **XNU**, которое расшифровывается как **X is Not Unix** (ироничное название, хотя на самом деле оно основано на **Unix**). Это гибридное ядро, сочетающее в себе элементы микроядра и монолитного ядра.

**Кратко:**

* + - 1. **Windows** — для ПК и ноутбуков.
      2. **Linux** — для серверов и разработки.
      3. **macOS** — для компьютеров Apple.
      4. **Android** — для смартфонов и планшетов.
      5. **iOS** — для iPhone и iPad.

****

**Как работает аперационная система:**

1. **Запуск железа**:

- Когда вы включаете компьютер, ОС загружается первой (это называется **загрузка системы**).

- ОС проверяет все аппаратные компоненты (процессор, память, жесткий диск, видеокарту и т.д.) и инициализирует их для работы.

1. **Организация работы**:

- ОС управляет ресурсами компьютера (процессорное время, оперативная память, дисковое пространство) и распределяет их между программами.

- Например, если вы запускаете браузер и текстовый редактор одновременно, ОС решает, сколько ресурсов выделить каждому приложению.

1. **Выполнение задач**:

- ОС предоставляет интерфейс (графический или командный), через который вы взаимодействуете с компьютером.

- Вы запускаете программы (например, браузер, текстовый редактор, игры), и ОС обеспечивает их работу, используя ресурсы железа.

1. **Управление данными**:

- ОС отвечает за хранение и доступ к данным на жестком диске или SSD.

- Когда вы сохраняете файл, ОС записывает его на диск, а когда открываете — считывает.

1. **Обеспечение безопасности**:

- ОС защищает ваш компьютер от вредоносных программ и несанкционированного доступа.

Без операционной системы ваш компьютер был бы просто набором "мертвого" железа. ОС — это то, что делает его функциональным и позволяет вам выполнять задачи: работать, учиться, играть, общаться и многое другое.

**Для чего тестировщику это знать?**

1. Программы разработанные для конкретной OS будут работать только на ней.
2. Тестировщики должны учитывать обновление OS и тестировать на разных версиях OS продукт.
3. Если мы идем на проект, то нужно сразу узнавать для какой OS будет разрабатываться продукт.
4. Разные OS предоставляют разные API для доступа к функциям устройства (камера, GPS, уведомления и т.д).

**API (Application Programming Interface)** — это набор правил, протоколов и инструментов, которые позволяют разным программам взаимодействовать друг с другом. Если говорить кратко:

**API** — это как "посредник" между двумя программами. Он позволяет одной программе запрашивать данные или функции у другой программы, не зная, как она устроена внутри.

**Пример из жизни:**

Представьте, что вы в ресторане. Вы (клиент) делаете заказ официанту (API), а официант передает ваш заказ на кухню (сервер). Вам не нужно знать, как готовится блюдо, — вы просто получаете результат.

**Что такое интернет?**

**Интернет** — это глобальная сеть, которая объединяет миллиарды устройств (компьютеры, серверы, смартфоны, планшеты и другие гаджеты) по всему миру. Он позволяет обмениваться данными, получать доступ к информации, общаться, работать, учиться и развлекаться.

1. **Сеть сетей**: Он состоит из множества локальных и глобальных сетей, соединенных между собой.
2. **Инфраструктура**: Кабели (оптоволокно, подводные кабели), спутники, серверы и маршрутизаторы.
3. **Протоколы**: Правила, по которым устройства обмениваются данными (например, TCP/IP, HTTP, FTP).

**Как работает интернет?**

1. **Устройства подключаются к сети**:

**-** Через Wi-Fi, Ethernet, мобильную связь (4G/5G),ройтеры, модемы и т.д.

1. **Данные передаются через серверы**:
   * + Когда вы открываете сайт, ваш запрос идет на сервер, который хранит данные этого сайта.
2. **Информация возвращается к вам**:
   * + Сервер отправляет данные (например, HTML-страницу) обратно на ваше устройство.



**Основные компоненты интернета**

1. **Устройства пользователей**:
   * + Компьютеры, смартфоны, планшеты, умные устройства.
2. **Серверы**:

**-** Хранят данные, обрабатывают запросы и отпраляют данные(например, веб-сайты, облачные сервисы).



1. **Провайдеры**:
   * + Компании, которые предоставляют доступ к интернету (например, через кабель или мобильную связь).
2. **Протоколы** (это набор правил как данные должны передаваться в интернете):
   * + **TCP/IP:** Основной протокол передачи данных.
     + **HTTP/HTTPS:** Для доступа к веб-сайтам.
     + **FTP:** Для передачи файлов.

**Internet Protocol**

**IP (Internet Protocol)** — это правила для передачи данных в интернете.  
**IP-адрес** — уникальный номер устройства в сети (например, 192.168.1.1).  
Нужен для:

1. Идентификации устройств.
2. Передачи данных (например, открытие сайтов).

Есть **внутренние IP** (для локальных сетей) и **внешние IP** (для интернета).

**HTTP/HTTPS**

**HTTP (HyperText Transfer Protocol):**

* + - 1. **Что это**: Протокол для передачи данных в интернете (например, веб-страниц).
      2. **Безопасность**: Данные передаются **в открытом виде** (не зашифрованы).
      3. **Пример**: <http://example.com>.

**HTTPS (HyperText Transfer Protocol Secure):**

**1. Что это**: Безопасная версия HTTP с шифрованием данных.

**2. Безопасность**: Данные передаются **зашифрованными** с использованием SSL/TLS.

**3. Пример**: <https://example.com>.

**Разница:**

1. **Шифрование**:
   * + **HTTP:** данные передаются открыто (можно перехватить).
     + **HTTPS:** данные зашифрованы (безопасно).
2. **Порт**:
   * + **HTTP использует порт 80.**
     + **HTTPS использует порт 443.**
3. **Сертификаты**:
   * + **HTTPS требует SSL/TLS-сертификат для шифрования.**

**FTP (File Transfer Protocol)**

**FTP (File Transfer Protocol)** — это сетевой протокол для передачи файлов между компьютерами в сети. Он позволяет загружать, скачивать, удалять и управлять файлами на удалённом сервере. FTP использует два канала: один для данных (передача файлов), другой для команд (управление соединением). Обычно работает на портах 20 и 21.

**Основные особенности:**

* + - Простота в использовании.
    - Поддержка аутентификации (логин/пароль).
    - Возможность работы в активном и пассивном режимах.

Однако FTP не шифрует данные, что делает его уязвимым для атак. Для безопасной передачи файлов рекомендуется использовать SFTP (SSH File Transfer Protocol) или FTPS (FTP Secure).

**Приложения и их разновидности**

* + - 1. **Веб-приложения (Web Applications)**
    1. **Веб-сайты (Web Sites):**
    - Это статические или динамические страницы, которые отображаются в браузере. Они могут содержать информацию, но не всегда имеют сложную функциональность.
    - Пример: блоги, новостные сайты, landing pages.

**Landing Page** (целевая страница) - это отдельная веб-страница, созданная специально для маркетинговых или рекламных целей. Её задача — побудить пользователя выполнить конкретное действие

* + 1. **Веб-приложения (Web Applications):**
    - Это более сложные приложения, которые работают в браузере и взаимодействуют с backend (серверной частью). Они предоставляют пользователю возможность выполнять различные действия (например, онлайн-банкинг, социальные сети, почтовые сервисы).
    - Пример: Gmail, Facebook, Google Docs.

A green and orange circles with black text

AI-generated content may be incorrect.

* + 1. **Веб-сервисы (Web Services):**
    - Это backend-компоненты, которые обеспечивают взаимодействие между различными системами через API (Application Programming Interface). Они не имеют пользовательского интерфейса, но предоставляют данные и функциональность другим приложениям.
    - Пример: REST API, SOAP API.

**Web Service** - это как "посредник" между двумя программами или приложениями. Он позволяет им обмениваться данными через интернет, даже если они написаны на разных языках программирования или работают на разных платформах.

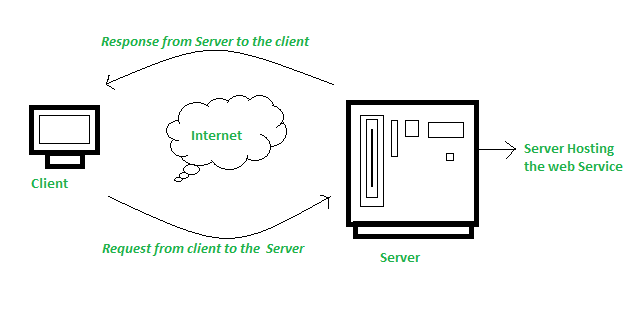
**Основные типы веб-сервисов:**

* + - * 1. **SOAP (Simple Object Access Protocol):**

Старый, но надежный стандарт. Использует XML для передачи данных. Часто используется в корпоративных системах.

* + 1. **REST (Representational State Transfer):**

Современный и популярный подход. Использует JSON (легкий формат данных) и работает через HTTP. Большинство современных API (например, Twitter, Google Maps) используют REST.

****

**Как Web Service связано с тестированием?**

Тестировщик должен проверять, как веб-сервис:

* Принимает запросы.
* Обрабатывает их.
* Возвращает корректные ответы.
* Работает при большой нагрузке (например, если одновременно много пользователей отправляют запросы).

1. **Десктопные приложения (Desktop Applications)**
   * + Это приложения, которые устанавливаются на компьютеры и работают под управлением операционной системы (Windows, macOS, Linux).
     + Они могут быть как standalone (работают без подключения к интернету), так и клиент-серверными (взаимодействуют с сервером).
     + Пример: Microsoft Office, Adobe Photoshop, Discord.
2. **Мобильные приложения (Mobile Applications)**
   * + - 1. **Нативные приложения (Native Applications):**
     + Разработаны специально для конкретной операционной системы (iOS, Android) с использованием языков программирования, поддерживаемых этой ОС (Swift/Objective-C для iOS, Kotlin/Java для Android).
     + Имеют доступ к аппаратным функциям устройства (камера, GPS, микрофон и т.д.).
     + Пример: Instagram, WhatsApp.

**Cross-platform native appliccations** — это отдельная категория. Они разрабатываются с использованием фреймворков, таких как **Flutter** или **React Native**, которые позволяют писать один код для обеих платформ (Android и iOS), но при этом приложения остаются нативными (используют нативные компоненты ОС).

* 1. **Мобильные веб-приложения (Mobile Web Applications):**
     + Это веб-сайты или веб-приложения, адаптированные для отображения на мобильных устройствах. Они работают через браузер и не требуют установки.
     + Пример: мобильная версия сайта Amazon или Wikipedia.
  2. **Гибридные приложения (Hybrid Applications):**
     + Это комбинация нативных и веб-приложений. Они разрабатываются с использованием веб-технологий (HTML, CSS, JavaScript), но упаковываются в нативное приложение, что позволяет им работать как нативные приложения и иметь доступ к аппаратным функциям устройства.
     + Пример: приложения, созданные с помощью Ionic или React Native.

**Гибридное приложение** - это приложение, которое выглядит и работает как веб-сайт, но при этом оно **установлено на устройстве** и может использовать функции смартфона или планшета (камера, GPS, уведомления и т.д.).



**Что такое браузер?**

**Веб-браузер(Web Browser)** — это программа для просмотра веб-страниц, поиска информации, обмена данными и взаимодействия с интернетом. Он позволяет открывать сайты, скачивать файлы, воспроизводить видео и многое другое.

**Основные браузеры и их отличия:**

1. **Google Chrome**
   * + **Разработчик**: Google.
     + Быстрый и производительный.
     + Множество расширений и интеграция с сервисами Google.
     + Высокое потребление оперативной памяти.
2. **Mozilla Firefox**
   * + **Разработчик**: Mozilla Foundation.
     + Акцент на приватность и защиту данных.
     + Гибкость в настройках и открытый исходный код.
     + Меньше расширений, чем у Chrome, но выше уровень безопасности.
3. **Safari**
   * + **Разработчик**: Apple.
     + Оптимизирован для устройств Apple.
     + Энергоэффективный с хорошей защитой приватности.
     + Лучше всего работает в экосистеме Apple.
4. **Microsoft Edge**
   * + **Разработчик**: Microsoft.
     + Преемник Internet Explorer, работает на движке Chromium.
     + Хорошо интегрирован с Windows и сервисами Microsoft.
     + Поддержка расширений Chrome и встроенные инструменты для продуктивности.
5. **Opera**
   * + **Разработчик**: Opera Software.
     + Встроенный VPN для конфиденциальности и доступа к заблокированным сайтам.
     + Блокировщик рекламы и режим экономии трафика.
     + Интеграция мессенджеров и низкое потребление ресурсов.
     + Подходит для пользователей, ценящих дополнительные функции и оптимизацию.

Для тестировщика важно учитывать **статистику популярности браузеров** и **устройств** (гаджетов или компьютеров), чтобы предлагать оптимальные решения для тестирования продукта. Вот пошаговый подход, как это можно сделать:

**1. Анализ статистики браузеров**

1. Использовать данные из авторитетных источников, таких как:
   * + [StatCounter](https://gs.statcounter.com/)
     + [NetMarketShare](https://www.netmarketshare.com/)
     + [W3Counter](https://www.w3counter.com/).
2. Обратить внимание на:
   * + **Популярные браузеры** (например, Google Chrome, Safari, Mozilla Firefox, Microsoft Edge).
     + **Их версии** (например, Chrome 120, Safari 17).
     + **Доля на рынке** в зависимости от региона (например, Safari популярен на iOS-устройствах в США и Европе).

**2. Анализ устройств**

1. Определить, на каких устройствах используется продукт:
   * + **Десктопы** (Windows, macOS, Linux).
     + **Мобильные устройства** (iOS, Android).
     + **Планшеты**.
2. Учитывать:
   * + Разрешения экранов (например, 1920x1080 для десктопов, 375x812 для iPhone 13).
     + Типы устройств (например, iPhone, Samsung Galaxy, iPad).

**3. Приоритизация тестирования**

1. На основе статистики выделить **наиболее популярные комбинации** браузеров и устройств. Например:
   * + Chrome на Windows (десктоп).
     + Safari на iPhone (мобильное устройство).
     + Chrome на Android (мобильное устройство).
2. Сосредоточиться на тестировании этих комбинаций в первую очередь.

**4. Предложение оптимального решения**

1. На основе анализа предлогать:
   * + **Список браузеров и устройств** для тестирования.
     + **Приоритеты** (например, сначала тестируем Chrome и Safari, затем Firefox и Edge).
     + **Рекомендации по кросс-браузерной совместимости** (например, использование инструментов вроде BrowserStack или Sauce Labs для тестирования на разных платформах).

**5. Пример решения**

1. Если продукт разрабатывается для **мобильных устройств**, и статистика показывает, что:
   * + 60% пользователей используют **Safari на iOS**,
     + 30% **Chrome на Android**,
     + 10% — другие браузеры,
2. то предложить:
   * + Основное внимание уделить тестированию на **Safari (iOS)** и **Chrome (Android)**.
     + Проверить поддержку на других браузерах (например, Firefox, Edge) только после покрытия основных сценариев.

**Список функций работы с вкладками в Google Chrome**

**1. Группировка вкладок**

- Возможность объединять вкладки в группы для удобства навигации.

- Полезно для организации работы с множеством открытых страниц.

**2. Командная строка (омнибокс)**

- Работает как поисковая строка и поддерживает **автодополнение (auto-suggest)**.

- Позволяет быстро находить сайты, выполнять поиск и вводить команды.

**3. Закрепление вкладок**

- Вкладку можно **закрепить**, чтобы она оставалась открытой при следующем запуске браузера.

- Удобно для часто используемых страниц.

**4. Закладки**

- Добавление страницы в закладки: нажать на **панель страницы** и выбрать **"Add page"**.

- Полезно для быстрого доступа к важным сайтам.

**5. История браузера**

- Полезно обращаться к истории, если случайно закрыли нужную вкладку.

- Быстрый доступ через **Ctrl+H** (Windows) или **Cmd+Y** (Mac).

**6. Менеджер паролей**

- Возможность просматривать сохраненные пароли через **Password Manager**.

- Удобно для управления учетными данными.

**7. Система безопасности**

- Браузер может блокировать доступ к подозрительным сайтам.

- Это можно обойти, принудительно разрешив открытие сайта (но с осторожностью).

**8. Автопереводчик страниц**

**- Не рекомендуется** использовать функцию авто перевода, так как:

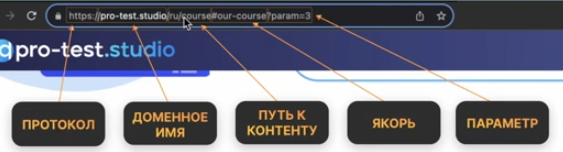
* + - Переводчик может вводить в заблуждение.
    - Не всегда работает корректно, что может быть ошибочно принято за баг.

**URL (Uniform Resource Locator)**

**URL (Единый указатель ресурсов)** — это адрес веб-страницы или ресурса, который находится в интернет-пространстве.

Ссылка состоит из нескольких частей:

1. **Протокол**
   * + Определяет способ передачи данных.
     + Примеры: HTTP (незащищенный) или HTTPS (защищенный).
2. **Доменное имя**
   * + Уникальное название ресурса (например, google.com).
3. **Путь к контенту**
   * + Указывает место на сервере, где хранится информация (например, /search).
4. **Якорь (anchor)**
   * + Часть после символа #, которая указывает на конкретное место на странице (например, #section).
5. **Параметры**
   * + Данные, которые передаются после символа ?.
     + Важная часть ссылки, которая передает дополнительные данные на сервер.
     + Они используются для фильтрации, поиска, сортировки и других операций на веб-страницах.
     + Параметры разделяются амперсандом & (например, ?q=test&page=1).



**Типы ссылок:**

Существует два вида ссылок:

1. **Абсолютный URL (Absolute URL)**
   * + Полный путь к ресурсу, включая протокол, доменное имя, путь и параметры.
     + Пример: https://example.com/page?q=test#section.
2. **Относительный URL (Relative URL)**
   * + Часть пути, которая используется в контексте текущей страницы.
     + Пример: /page или ../images/photo.jpg.

**URL (Uniform Resource Locator)**

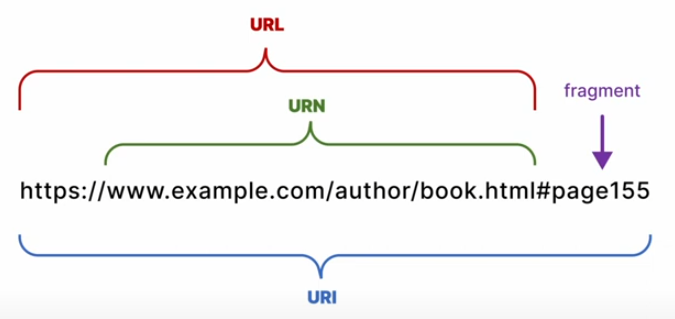
* **Что это**: Адрес/ссылка, которая указывает, **где находится ресурс**.
* **Особенность**: Всегда включает **протокол** (например, http:// или https://).
* **Пример**: https://example.com/page.

**URI (Uniform Resource Identifier)**

* **Что это**: Общий термин для идентификации ресурса.
* **Особенность**: Может включать как **URL**, так и **URN**.
* **Показывает**: Всю ссылку от начала до конца.
* **Пример**: https://example.com/page#section или urn:isbn:0451450523.

**URN (Uniform Resource Name)**

* **Что это**: Уникальное имя ресурса.
* **Особенность**: **Не зависит от местоположения** и не включает протокол.
* **Пример**: urn:isbn:0451450523 (уникальный идентификатор книги).



**Плагины (или расширения) для браузера**

**Плагины (или расширения)** — это небольшие программы, которые интегрируются в браузер и добавляют новые возможности.

Они могут:

* + - Блокировать рекламу.
    - Улучшать безопасность.
    - Добавлять новые инструменты для разработчиков.
    - Изменять внешний вид сайтов.
    - Упрощать выполнение повседневных задач.

**Интерфейсы (Interfaces): GUI, CLI, API**

A screenshot of a computer

AI-generated content may be incorrect.

**GUI (Graphical User Interface) — Графический интерфейс пользователя**

**Описание**: Это визуальный интерфейс, который позволяет пользователям взаимодействовать с программой или устройством с помощью графических элементов, таких как кнопки, меню, окна, иконки и т.д.

**Как используется**: Управление осуществляется с помощью мыши, клавиатуры, сенсорного экрана или других устройств ввода.

**Примеры**:

* + - Операционные системы (Windows, macOS, Linux с графической оболочкой).
    - Веб-браузеры (Google Chrome, Firefox).
    - Мобильные приложения (Instagram, WhatsApp).

**A screenshot of a computer

AI-generated content may be incorrect.**

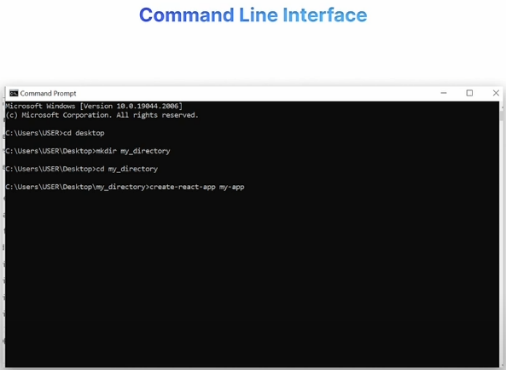
**CLI (Command Line Interface) — Интерфейс командной строки**

**Описание**: Это текстовый интерфейс, где пользователь взаимодействует с системой(OS,Applications) путем ввода команд с клавиатуры.

**Как используется**: Пользователь вводит команды в текстовом формате, а система выполняет их и выводит результат.

**Примеры**:

* + - Терминал в Linux или macOS.
    - Командная строка Windows (cmd, PowerShell).
    - Утилиты для администрирования (ping, ssh, git).



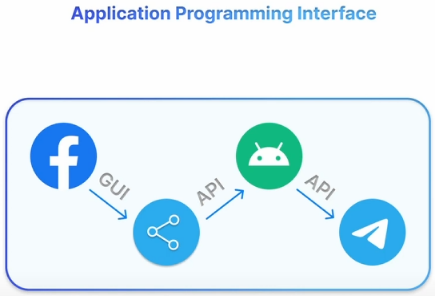
**API (Application Programming Interface) — Интерфейс программирования приложений**

**Описание**: Это набор правил, протоколов и инструментов, которые позволяют программам взаимодействовать друг с другом. API не предназначен для прямого взаимодействия с пользователем, а скорее для взаимодействия между программами.

**Как используется**: Разработчики используют API для интеграции функций одной программы или сервиса в другую. Например, веб-сайт может использовать API для получения данных от стороннего сервиса (например, погода, курсы валют).

**Примеры**:

* + - Веб-API (Google Maps API, Twitter API).
    - Библиотеки для программирования (например, API для работы с базами данных).
    - API операционных систем (например, WinAPI в Windows).



**Основные разновидности API**

* + - 1. **По уровню доступа и использования (By Access Level):**

**Публичные (Public API or Open API)**:

* + - Доступны для всех разработчиков.
    - Пример: Google Maps API, Twitter API.
    - Часто используются для интеграции сторонних сервисов.

**Частные (Private API)**:

* + - Используются внутри компании или организации.
    - Не доступны для внешних разработчиков.
    - Пример: внутренние API для взаимодействия между микросервисами в компании.

**Партнерские (Partner API)**:

* + - Доступны только для определенных партнеров или сторонних разработчиков по соглашению.
    - Пример: API для интеграции с платежными системами (например, Stripe, PayPal).

**2. По архитектуре и протоколам (By Architecture and Protocols):**

**REST (Representational State Transfer)**:

* + - Самый популярный тип API.
    - Использует HTTP-запросы (GET, POST, PUT, DELETE).
    - Данные передаются в формате JSON или XML.
    - Пример: API социальных сетей (Facebook, Instagram).

**SOAP (Simple Object Access Protocol)**:

* + - Более старый и строгий протокол.
    - Использует XML для передачи данных.
    - Подходит для сложных enterprise-систем.
    - Пример: API для банковских транзакций.

**GraphQL**:

* + - Позволяет клиенту запрашивать только нужные данные.
    - Более гибкий, чем REST.
    - Пример: API GitHub, Shopify.

**gRPC API (Google Remote Procedure Call)**

* + - Использует протокол HTTP/2 и бинарный формат данных.
    - Подходит для высоконагруженных систем.
    - Пример: микросервисы в облачных приложениях.

**3. По назначению** **(By Purpose):**

**Веб-API (Web API)**:

* + - Используются для взаимодействия с веб-сервисами через HTTP.
    - Пример: API для получения данных о погоде (OpenWeatherMap).

**API операционных систем (OS API)**:

* + - Предоставляют доступ к функциям ОС.
    - Пример: WinAPI (Windows), POSIX API (Linux).

**Библиотеки и фреймворки (Library/Framework API)**:

* + - API, встроенные в языки программирования или библиотеки.
    - Пример: API для работы с базами данных (SQLAlchemy для Python).

**Аппаратные API (Hardware API)**:

* + - Используются для взаимодействия с оборудованием.
    - Пример: API для управления принтерами, камерами.

**4. По способу реализации (By Implementation):**

**Синхронные API (Synchronous API)**:

* + - Клиент ждет ответа от сервера перед продолжением работы.
    - Пример: большинство REST API.

**Асинхронные API (Asynchronous API)**:

* + - Клиент не ждет ответа и может продолжать работу.
    - Пример: API для обработки длительных задач (отправка email).

**5. По типу данных (By Data Format):**

**JSON API**:

* + - Данные передаются в формате JSON.
    - Пример: большинство современных REST API.

**XML API**:

* + - Данные передаются в формате XML.
    - Пример: SOAP API.

**Бинарные API (Binary API)**:

* + - Данные передаются в бинарном формате.
    - Пример: gRPC.

**6. По области применения (By Application Domain):**

**Социальные сети (Social Media API)**:

* + - Пример: Facebook API, Twitter API.

**Платежные системы (Payment API)**:

* + - Пример: Stripe API, PayPal API.

**Картографические сервисы (Maps API)**:

* + - Пример: Google Maps API, Yandex Maps API.

**Облачные сервисы (Cloud API)**:

* + - Пример: AWS API, Google Cloud API.

**Введение в тестирование**

**Тестирование** — это процесс проверки программного обеспечения (или другого продукта) на соответствие ожидаемым требованиям, а также выявление дефектов и отклонений от ожидаемого поведения.

Тестирование включает в себя:

1. **Сравнение ожидаемого и фактического результата**: Тестирование позволяет определить, соответствует ли продукт заданным требованиям и ожиданиям пользователей. Если фактический результат отличается от ожидаемого, это может указывать на наличие дефекта.
2. **Выявление дефектов**: Дефекты (баги) — это отклонения от ожидаемого поведения продукта. Тестирование помогает обнаружить эти отклонения, чтобы их можно было исправить.
3. **Оценка качества продукта**: Тестирование не только выявляет дефекты, но и помогает оценить общее качество продукта, его надежность, производительность, удобство использования и другие характеристики.
4. **Предотвращение проблем**: Тестирование также может быть направлено на предотвращение потенциальных проблем, которые могут возникнуть в будущем при использовании продукта.

A diagram of a company

AI-generated content may be incorrect.

**Навыки для тестировщика**

**Hard Skills (технические навыки):**

1. Знание основ тестирования (виды, методы, техники).
2. Знание английского языка.
3. Основы программирования (для автоматизации: Python, Java, JavaScript и др.).
4. Знание веб-тезнологий, приложений, геймдева.
5. Работа с баг-трекинговыми системами (Jira, Trello и др.).
6. Понимание SQL и баз данных.
7. Работа с инструментами автоматизации (Selenium, Postman, SoapUI и др.).
8. Знание API-тестирования (REST, SOAP).
9. Понимание клиент-серверной архитектуры.

**Soft Skills (гибкие навыки):**

1. Внимательность к деталям.
2. Усидчивость.
3. Умение учиться и адаптироваться.
4. Коммуникабельность (работа с командой).
5. Ответственость+Тайм-менеджмент.
6. Аналитическое мышление.
7. Критическое мышление (поиск неочевидных проблем).

A screen shot of a computer

AI-generated content may be incorrect.

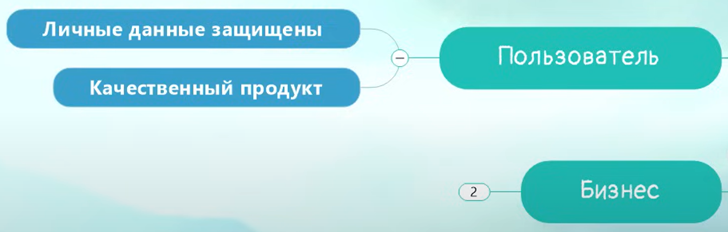
**Для чего необходимо тестирование?**

**Пользователю:**

1. **Качественный продукт**: Убедиться, что продукт работает корректно, без сбоев и ошибок.
2. **Безопасность**: Защита данных пользователя и предотвращение уязвимостей.
3. **Удобство использования**: Проверка, что интерфейс интуитивно понятен и удобен.
4. **Надежность**: Продукт должен стабильно работать в разных условиях.

**Бизнесу:**

1. **Снижение рисков**: Предотвращение убытков из-за багов или сбоев.
2. **Репутация**: Высокое качество продукта укрепляет доверие клиентов.
3. **Экономия**: Раннее выявление дефектов дешевле, чем их исправление после релиза.
4. **Конкурентоспособность**: Качественный продукт выделяется на рынке.
5. **Соответствие требованиям**: Выполнение стандартов и нормативов (например, GDPR, ISO).



**Обязанности тестировщика**

1. **Разработка тестовой документации:** тест-планы, тест-кейсы, чек-листы.
2. **Проведение тестирования:** ручное и/или автоматизированное, проверка функциональности, регрессионное тестирование.
3. **Оформление баг-репортов:** описание дефектов, приоритизация, ретест.
4. **Анализ и отчетность:** подготовка отчетов, коммуникация с командой.
5. **Работа с требованиями:** анализ и уточнение.
6. **Постоянное улучшение процессов:** изучение новых инструментов, оптимизация тестирования.



**SDLC - Software Development Life Cycle**

**SDLC** - это систематизированный процесс, этапы которого охватывают полный жизненный цикл программного обеспечения **(Software Lifecycle)** и который определяет различные этапы разработки программного обеспечения для создания высококачественного программного обеспечения, отвечающего ожиданиям клиентов и для улучшения эффективности разработки. Разработка системы должна быть завершена в заранее определенные сроки и стоимость. Каждая фаза жизненного цикла SDLC имеет свой собственный процесс и результаты, которые используются в следующей фазе.

Обычно он делится на шесть-восемь шагов, но менеджеры проектов могут объединять, декомпозировать или пропускать шаги, в зависимости от скоупа проекта.

В разных источниках фазы немного отличаются, но глобально суть везде одинакова.

**Фазы SDLC**

A diagram of a circular diagram

AI-generated content may be incorrect.

1. **Сбор и анализ требований** **(Requirement Gathering and Analysis):** На этом этапе от клиентов собирается вся необходимая информация для разработки продукта в соответствии с их ожиданиями. Любые неясности должны быть разрешены сразу на этом этапе. Бизнес-аналитик и менеджер проекта назначили встречу с заказчиком, чтобы собрать всю информацию, например, что заказчик хочет построить, кто будет конечным пользователем, какова цель продукта. Перед созданием продукта очень важно понимание или знание продукта. Например, клиент хочет иметь приложение, которое включает денежные транзакции. В этом случае требование должно быть четким, например, какие транзакции будут выполняться, как они будут проводиться, в какой валюте они будут проводиться и т. д. После того, как сбор требований завершен, проводится анализ для проверки возможности разработки продукта. После четкого понимания требования создается документ **SRS (Software Requirements Specification).** Этот документ должен быть полностью понятен разработчикам, а также должен быть рассмотрен заказчиком для использования в будущем.

A screenshot of a computer

AI-generated content may be incorrect.

1. **Дизайн (Design):** На этом этапе требования, собранные в документе SRS, используются в качестве входных данных, и создается архитектура программного обеспечения, которая используется для реализации разработки системы. Подключается UI/UX.
2. **Осуществление и разработка (Implementation or Coding):** Реализация / кодирование начинается, как только разработчик получает Design document. Дизайн программного обеспечения переведен в исходный код. На этом этапе реализуются все компоненты программного обеспечения;
3. **Тестирование (Testing):** В классическом понимании, как фаза SDLC, тестирование начинается после завершения кодирования и выпуска модулей для тестирования. Однако, на самом деле тестирование присутствует практически на всех этапах SDLC, начиная с анализа собранных требований. На этом этапе разработанное программное обеспечение тщательно тестируется, и все обнаруженные дефекты передаются разработчикам для их исправления. Повторное тестирование, регрессионное тестирование проводится до тех пор, пока программное обеспечение не будет соответствовать ожиданиям клиента. Тестировщики обращаются к документу SRS, чтобы убедиться, что программное обеспечение соответствует стандарту заказчика;
4. **Развертывание (Deployment):** После тестирования продукта он развертывается в производственной среде или выполняется первое UAT (пользовательское приемочное тестирование), в зависимости от ожиданий клиента. В случае UAT создается копия производственной среды, и заказчик вместе с разработчиками выполняет тестирование. Если клиент остается доволен, то предоставляет согласие на релиз.

**UAT (User Acceptance Testing)** — это финальный этап тестирования, на котором **пользователи** (или заказчики) проверяют, соответствует ли продукт их требованиям и готов ли он к использованию в реальных условиях.

1. **Поддержка (Maintenance):** Основное внимание на этом этапе SDLC уделяется обеспечению того, чтобы потребности продолжали удовлетворяться и чтобы система продолжала работать в соответствии со спецификацией, упомянутой в первом этапе. После того, как система развернута и клиенты начинают использовать разработанную систему, следует 3 вида активностей:

**1) Исправление ошибок;**

**2) Обновление;**

**3) Улучшение.**

В некоторых источниках дополнительно выделяют следующие этапы:

1. **Идея, или планирование системы, который предшествует этапу "Анализ требований".** Представляет собой определение проблемы, которую поможет решить разрабатываемый продукт.
2. **Ввод в эксплуатацию** - синоним этапа "Развертывание".
3. **Вывод из эксплуатации, который завершает SDLC.** Например, при устаревании продукта и прекращении его поддержки, необходимо сформировать процесс вывода продукта из эксплуатации, который включает в себя: сохранение информации, удаление данных, удаление аппаратного и программного обеспечения

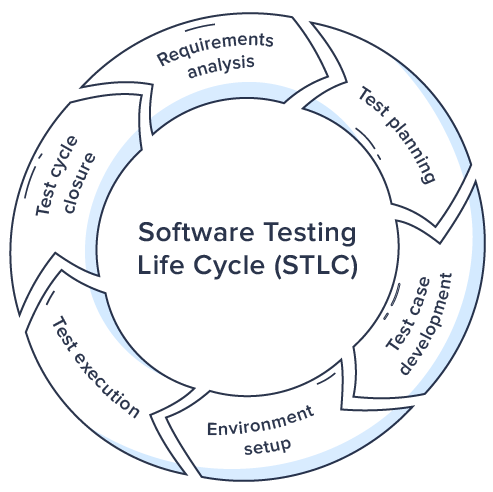
A screenshot of a computer

AI-generated content may be incorrect.

**STLC - Software Testing Lifecycle**

**STLC** (жизненный цикл тестирования ПО)- это процесс тестирования, который включает в себя определенную последовательность шагов, чтобы гарантировать достижение целей в области качества. В процессе STLC каждое действие выполняется планомерно и систематически. Каждый этап имеет разные цели и результаты. У разных организаций разные этапы STLC, однако основа остается прежней.

**Фазы STLC**



STLC имеет несколько взаимосвязанных фаз и в целом очень похож на SDLC. Эти фазы являются последовательными и называются:

* 1. **Анализ требований** **(Requirement Analysis):** один из важнейших этапов, потому что именно на нем можно почти бесплатно исправить недостатки проекта. Этап анализа требований также определяет потенциальную потребность в автоматизированном тестировании и позволяет производить экономические расчеты затрат на рабочую силу на основе оценки проекта. На этом же этапе обсуждаются и документируются критерии начала и окончания тестирования.
  2. **Планирование тестирования** **(Test Planning):** на этом этапе формируется план тестирования, т.е. мы определяем действия и ресурсы, которые помогут достичь целей тестирования (участники и их роли, инструменты, окружение). Во время планирования мы также пытаемся определить метрики, метод сбора и отслеживания этих метрик. План составляют исходя из требований, тестовой стратегии и анализа рисков.

**Метрики (metrics)** в контексте тестирования — **это количественные или качественные показатели**, которые помогают оценить процесс тестирования, качество продукта и эффективность работы команды. Они используются для анализа, контроля и улучшения процесса тестирования. Метрики позволяют принимать обоснованные решения на основе данных, а не на интуиции.

* 1. **Разработка тест-кейсов (Test Case Development):** подразумевает использование ручного и автоматизированного тестирования для достижения полного охвата функциональности программного обеспечения, при этом процесс основан на заранее установленных требованиях. Чаще всего тест-кейсы для автоматического тестирования пишутся отдельно, так как кейсы для ручного тестирования описаны в виде шпаргалок (cheat sheets)
  2. **Настройка тестовой среды (Test Environment Setup):** в плане тестирования четко указано, какую тестовую среду следует использовать. На этом этапе STLC настраиваются операционные системы и виртуальные машины, развертываются инструменты тестирования, такие как Selenium, Katalon Studio, а также тестовая среда и базы данных проекта. Мы также обращаемся с запросами к DevOps и администраторам, если требуется поддержка

1. **Выполнение тестов (Test Execution):** тесты выполняются на основе готовой тестовой документации и правильно настроенной тестовой среды. Все результаты тестирования регистрируются в Системе управления тестированием. Отрицательно пройденные тесты, в которых фактический результат отличается от ожидаемого, регистрируются как ошибки и передаются команде разработчиков на доработку с последующей перепроверкой после исправления
2. **Завершение цикла испытаний (Test Cycle Closure):** окончательная генерация отчетов о тестировании для клиента. Они должны включать затраченное время, процент обнаруженных ошибок и положительных результатов тестирования, общее количество обнаруженных и исправленных ошибок. Что касается отдела тестирования, то это момент для анализа его работы, подведения итогов, анализа его продуктивности и возможности внести предложения по улучшению качества тестирования

**Разница STLC и SDLC**

STLC и SDLC тесно связаны друг с другом, но они одновременно преследуют разные задачи с одной и той же целью, а именно:

* + сбор требований в желаемой форме и разработка заявленной функциональности (SDLC);
  + анализ требований, помощь клиенту и команде разработчиков и подтверждение качества реализованной функциональности (STLC).

**Общая цель - удовлетворение клиента и получение максимально возможного балла на этапах верификации и валидации.**