**Теория**

**1. IT-проект: основные понятия**

IT-проект - это проект, связанный с разработкой, внедрением или поддержкой информационных технологий.

В этом контексте можно рассмотреть такие термины, как цель проекта, задачи, сроки, бюджет, команда проекта, методология управления проектом и т.д.

**2. Компьютерный червь: свойства, особенности**

Компьютерный червь - это вредоносная программа, способная самостоятельно распространяться через компьютерные сети, в основном, путем использования уязвимостей.

Основные свойства и особенности компьютерных червей включают автономность, способность к самосохранению, способность к удаленному управлению и возможность нанесения вреда компьютерной системе, такой как уничтожение данных или рас

**3. Компьютерный троян: свойства, особенности.**

Компьютерный троян (Trojan) - это вид вредоносного программного обеспечения, которое маскируется под легитимное или полезное приложение или файл, чтобы получить доступ к компьютеру и нанести вред. В отличие от вирусов или червей, троянские программы не распространяются самостоятельно, они требуют участия пользователя для запуска.

Основные особенности и свойства компьютерных троянов:

1. Маскировка

2. Контроль над компьютером

3. Вредительство

4. Автоматическое распространение

**4. Компьютерный блокировщик экрана (WinLocker): свойства, особенности.**

Компьютерный блокировщик экрана, также известный как WinLocker, является видом вредоносного программного обеспечения, которое блокирует доступ к компьютеру и отображает на экране сообщение о требовании оплаты для разблокировки.

Основные свойства и особенности компьютерного блокировщика экрана:

1. Блокировка экрана

2. Вымогательство денег

3. Мошенничество

4. Распространение

5. Превращение в социальную инженерию

**5. Необходимые условия применения измерительных мониторов:**

1. Наличие готовой программы, подлежащей измерительному исследованию
2. Наличие реальной вычислительной системы, а не ее модели, для прогона программы
3. Наличие аппаратных или программных средств проведения измерений
4. Создание условий снижение искажений, вносимые в информат. систему до приемлемого уровня.

**6. Измерительные методы: назначение.**

В целом измерительные методы имеют следующее назначение:

1. Измерение параметров потребления программами ресурсов вычислительной системы с целью устранения дефектов производительности.

2. Предварительное измерение параметров системы для имитационных аналитических моделей программ перед их последующим использованием.

Это связано со сложностью оценки параметров моделей, особенно при использован новых системных средств с неизвестными динамическими параметрами (например устройств ввода - вывода нового процессора с не вполне известными характеристиками и т.п.).

3. Проверка адекватности имитационных или аналитических моделей и методов расчета характеристик выполненной программ по результатам моделирования

**7. Описание компонентов схемы измерений:**

1. Исследуемую вычислительную систему (ВС) с установленными программами.

2. Средства регистрации параметров потребляемых ресурсов при выполнении данной рабочей нагрузки.

3. Архив для хранения результатов многочисленных измерений.

4. Результаты измерений обрабатываются некоторой ВС (отдельная ВС или та же, на которой снимались измерения, но после выполнения сеанса измерений.).

5. Рабочая нагрузка - одна или несколько программ или наборов данных для получения статистики проводимых измерений.

**8. Аппаратные измерительные мониторы: виды, основные назначения.**

Аппаратные измерительные мониторы (АИМ).

АИМ подразделяются на встроенные и автономные.

Встроенные АИМ - включаются в аппаратуру системы заводом изготовителем, как правило, для выполнения тестовых измерений в фиксированном наборе внутренних точек устройств системы.

Основное назначение встроенного АИМ - для проверки, контроля и настройки ВС. Но т.к. в его составе могут находиться триттеры состояний схемы, счетчики и некоторые другие устройства, то они могут использоваться и для измерения параметров программ.

Автономные АИМ подключаются к измерительным точкам извне системы через специальные разъемы и могут задавать любые точки доступные для измерений.

Автономные АИМ наиболее общий случай.

**9. Измерительные мониторы** - это устройства или программное обеспечение, которые предназначены для измерения и мониторинга различных параметров или характеристик.

**10. Статическое тестирование** - Статическое тестирование - ручное тестирование программ, начиная со стадии формирования требований к программе. На стадии кодирования при статическом тестировании некоторую часть маршрутов исполнения тестируют вручную.

**11. Динамическое тестирование: общие сведения, значение.**

Динамическое тестирование подразумевает достаточно полную структурную и функциональную проверку выполнения программы.

Значение динамического тестирования заключается в том, что оно позволяет проверить работоспособность и функциональность программы в реальном времени. Путем выполнения тестовых сценариев можно выявить различные ошибки, такие как некорректное поведение программы, неработающие функции, проблемы с производительностью и т. д.

**12. Этапы формирования эталонов для тестирования и их описание.**

Существует несколько способов

формирования эталонов:

1. Использование аналитических выражений. Этот способ особенно подходит при детерминированном тестировании, так как имеется возможность сравнить результаты тестирования с ожидаемыми результатами. Имеются ограничения в использовании этого метода, если неизвестны или отсутствуют аналитические выражения связывающие входные данные и результаты; иногда требуется использовать много допущений.

2. Использование моделирования на ЭВМ. Способ является универсальным. При этом ряд данных моделируется другим способом и по другим алгоритмам, нежели испытываемая программа и на других ЭВМ Причем наборы входных данных создаются по случайным законам, что обеспечивает высокую гибкость этого способа.

3. Использование результатов испытаний предшествующих вариантов программ. При этом используется ранее накопленный опыт испытателя или других исследователей, выраженный в экспертных оценках ожидаемых результатов.

**13. Классификация анализа вредоносных программ:**

1) Статический анализ: включает в себя исследование кода и структуры программы без ее активного выполнения. Статический анализ используется для обнаружения уязвимостей, анализа потока данных и процедур, анализа паттернов злоупотребления, анализа API и других методов.

2) Динамический анализ: осуществляется путем активного выполнения вредоносной программы в контролируемой среде. Включает в себя анализ поведения программы, отслеживание системных вызовов, мониторинг сетевой активности, анализ деобфускации кода и прочее.

3) Анализ памяти: представляет собой изучение работы вредоносной программы в памяти компьютера, обнаружение модификации структур данных и раскрытие скрытых алгоритмов.

4) Анализ сетевой активности: изучение сетевого взаимодействия вредоносной программы с другими компьютерами или серверами. Помогает выявить команды, передаваемые управляющим серверам, внешние домены или IP-адреса связанные с злоумышленниками.

5) Обратное инженерия: заключается в исследовании исполняемого кода или программного обеспечения с целью выявления его работы и функций, включая анализ алгоритмов шифрования, методов скрытия и других инструментов.

**14. Этапы изучения вредоносного программного обеспечения и их описание:**

1) Сбор образца: на этом этапе вредоносная программа должна быть обнаружена и зафиксирована, чтобы можно было провести дальнейший анализ.

2) Декомпиляция и деобфускация: вредоносная программа может быть зашифрована, обфусцирована или скрыта. Этот этап включает раскрытие и восстановление исходного кода или структуры программы.

3) Анализ структуры и функций: на этом этапе исследуются различные модули, функции и алгоритмы вредоносной программы для понимания ее целей и возможностей.

4) Изучение сетевой активности: данный этап фокусируется на анализе сетевой активности вредоносной программы, поиск связанных доменов и IP-адресов, анализ протоколов и команд, передаваемых по сети.

5) Анализ поведения: вредоносная программа может иметь различные поведенческие характеристики, такие как манипуляция файлами, регистрацией клавиатуры или мониторингом сетевой активности. На этом этапе изучается поведение программы, чтобы понять ее функциональность.

6) Построение модели угрозы: на основе полученных данных и анализа строится модель угрозы, которая помогает оценить риски и разработать меры безопасности для предотвращения атак и защиты от вредоносных программ.

7) Создание отчета и рекомендаций: результаты анализа вредоносной программы документируются в отчете, который содержит описание ее характеристик, функциональности и рекомендации по защите от подобных атак в будущем.

**15. Основные инструменты для исследования кода программ:**

1. Интегрированная среда разработки (IDE) - это программное обеспечение, которое предоставляет разработчику удобную среду для написания, отладки и тестирования кода программы. Основные функции IDE включают автодополнение кода, отладку, интеграцию с системой контроля версий и анализ кода.

2. Отладчик (Debugger) - это инструмент, который позволяет разработчику выполнять код программы по шагам и проверять состояние переменных и объектов в процессе выполнения. Отладчик позволяет обнаруживать и исправлять ошибки в коде и понять, как программные объекты взаимодействуют друг с другом.

3. Профилировщик (Profiler) - это инструмент, который помогает оптимизировать производительность программы. Профилировщик анализирует время, затраченное на выполнение каждого сегмента кода, и помогает выявить медленные участки программы. Это позволяет разработчикам устранить узкие места и повысить эффективность кода.

4. Система контроля версий (Version Control System, VCS) - это инструмент, который позволяет разработчику отслеживать и управлять изменениями в коде программы. Система контроля версий позволяет сохранять предыдущие версии кода, работать над изменениями вместе с другими разработчиками и откатывать изменения при необходимости. Некоторые популярные системы контроля версий - Git, SVN и Mercurial.

**16. Файловый монитор (FileMonitor)** - это инструмент, который позволяет отслеживать изменения, происходящие в файлах и директориях операционной системы. Файловый монитор предоставляет возможность отслеживать создание, удаление, изменение файлов и директорий, а также получать уведомления об этих событиях.

**17. Методы, применяемые для обнаружения модифицированного кода. Преимущества и недостатки их применения.**

Для обнаружения модифицированного кода применяются различные методы, включая статический анализ, динамический анализ.

Статический анализ - это метод, основанный на анализе исходного кода программы, без ее реального выполнения. Он позволяет выявить изменения в программном коде, которые могут указывать на модификации или внесение вредоносного кода.

Преимущества статического анализа включают его способность обнаружить различные типы изменений в коде и внести минимальные изменения в программу.

Однако недостатками статического анализа являются высокие требования к экспертным знаниям и сложности в обнаружении некоторых типов вредоносного кода.

Динамический анализ - это метод, основанный на исполнении программы и наблюдении за поведением кода в реальном времени. Он может выявить изменения в программе, которые проявляются только при выполнении кода.

Преимущества динамического анализа включают его способность обнаруживать изменения в программе в режиме реального времени и более низкие требования к экспертным знаниям.

Однако недостатками динамического анализа являются его сложность в обнаружении некоторых типов изменений в программе и возможные негативные эффекты на работу системы.

**18. Отладчик** - программа, предназначенная для анализа поведения другой программы, обеспечивающая остановку в указанных точках и позволяющая просматривать (редактировать) содержимое ячеек памяти, регистров процессора и команды программы.

**19. Трассировка кода программы** - это отладочный механизм, который позволяет программисту следить за выполнением программного кода и анализировать его работу поэтапно, пошагово.

**20. Контрольные точки останова** - это отладочный механизм, который позволяет программисту временно останавливать выполнение программы в определенных местах, чтобы проанализировать текущее состояние программы и переменных.

Идея механизма заключается во внесении в программный код специального однобайтового кода (0хСС) - так называемой контрольной точки останова.

**21. Понятие тестирования программ** - процесс исследования, испытания программного продукта(ПП), имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом ([ISO](https://ru.wikipedia.org/wiki/ISO)/IEC TR 19759:2005)

**22. Понятие анализа программного кода** означает изучение и оценку исходного кода программы для выявления ошибок, уязвимостей или других проблем, которые могут влиять на работу программы.

**23. Понятие и основные отличия тестирования моделей.**

Тестирование моделей – это процесс проверки моделей на соответствие требованиям и обнаружение возможных ошибок или дефектов.

Тестирование представляет собой деятельность по проверке программного кода и документации. Она должна заранее планироваться и систематически проводиться специально назначенным независимым тестировщиком. Работа тестировщика начинается до утверждения спецификаций требований. Он проверяет требования к ПП на полноту и возможность тестирования, определяет методы тестирования:

1. Модульное (осуществляет сам разработчик на этапе разработки);
2. Интеграционное (отвечает независимый тестировщик);
3. Системное (отвечает независимый тестировщик);
4. Выходное (отвечает независимый тестировщик);
5. Приемочное (выполняется совместно с представителями заказчика).

Тестирование 1-4 уровней проводится внутри организации.

**24. Характеристика и описание процесса тестирования методом "черный** **ящик".**

Метод "черный ящик" (black box testing) в тестировании – это подход, при котором тестирующий имеет доступ только к входным данным и ожидаемому результату, а внутреннее устройство и работа тестируемой системы остаются неизвестными.

**25. Метод "стеклянный ящик" (также известный как метод "белого ящика")** это метод тестирования программного обеспечения, который основывается на изучении внутренней структуры и логики программы. В этом методе тестировщик имеет доступ к исходному коду программы и может использовать его для разработки тестовых случаев и оценки покрытия кода.

Основные принципы метода "стеклянный ящик":

1. Анализ кода: тестировщик изучает исходный код программы, а также алгоритмы и структуры данных, используемые в программе. Это позволяет ему определить потенциальные проблемы и разработать тестовые случаи, которые позволяют проверить все возможные пути выполнения программы.

2. Покрытие кода: тестировщик стремится достичь максимального покрытия кода, то есть выполнить тесты, которые охватывают все строки кода, включая различные условия и ветвления. Это позволяет обнаружить потенциальные ошибки и улучшить качество программы.

3. Тестовая документация: тестировщик разрабатывает документацию, описывающую тестовые случаи, используемые данные и ожидаемые результаты. Это позволяет другим членам команды разработчиков и тестировщиков понять, какие аспекты программы были протестированы и какие проблемы были обнаружены.

4. Методы тестирования: тестировщик может использовать различные методы тестирования, такие как тестирование единичных компонентов (unit testing), тестирование интеграции (integration testing) и тестирование системы в целом (system testing). Все эти методы могут быть применены с использованием метода "стеклянный ящик".

Метод "стеклянный ящик" позволяет тестировщику получить глубокое понимание работы программы и обнаружить потенциальные проблемы, которые могут быть пропущены при других методах тестирования. Однако для его применения требуется доступ к исходному коду программы и достаточные знания программирования для анализа и разработки тестовых случаев.

**26. Эмулирующий отладчик** - отладчик, который самостоятельно интерпретирует и выполняет команды программы (без использования реального процессора).

**27. Методы проникновения вредоносного программного обеспечения на персональный компьютер могут включать следующие способы:**

1. Фишинг: Злоумышленники могут отправлять поддельные электронные письма или создавать поддельные веб-сайты, чтобы обмануть пользователей и заставить их предоставить личную информацию или установить вредоносное ПО.

2. Социальная инженерия: Злоумышленники могут попытаться обмануть пользователей с помощью манипуляций и убедительных методов, чтобы убедить их установить или открыть вредоносный файл или ссылку.

3. Эксплойты: Злоумышленники могут использовать уязвимости в операционной системе или программном обеспечении для внедрения вредоносного ПО. Эксплойты могут быть внедрены через вредоносные веб-сайты, почтовые вложения или даже через непатченые программы.

4. Вредоносные вложения электронной почты: Злоумышленники могут включать вредоносные программы во вложения электронной почты и отправлять их пользователям. Если пользователи открывают вложения, вредоносное ПО может установиться на их компьютерах.

5. Зараженные веб-сайты: Злоумышленники могут создавать или загружать вредоносное ПО на веб-сайты, которые пользователи посещают. Когда пользователи посещают такие сайты, вредоносное ПО может загрузиться и установиться на их компьютерах.

**28. Вредоносное программное обеспечение (ВПО)** – это программы, созданные с целью нанести вред компьютеру и пользователям. Оно может быть разработано для сбора важной информации, нарушения безопасности, уничтожения данных или похищения личных данных. Вредоносное ПО часто распространяется через поддельные электронные письма, веб-сайты, социальные сети или зараженные приставки USB. Оно может проникнуть на компьютер и выполняться в фоновом режиме, без ведома и согласия пользователя. Примеры вредоносного ПО включают вирусы, черви, трояны, шпионское ПО и рекламное ПО. Понимание понятия вредоносного ПО позволяет пользователям принимать меры для защиты своих компьютеров и важной информации.

**29. Понятие метрики измерений:**

Метрика измерений – это средство для оценки и измерения определенной характеристики или параметра в рамках определенной системы или процесса. Она может быть количественной или качественной и используется для оценки эффективности, производительности или качества данной системы или процесса. Метрики измерений помогают собирать, анализировать и интерпретировать данные для принятия решений и оптимизации процессов.

**30. Хранилище (Repository): основные понятия:**

Хранилище (Repository) – это централизованное место, где хранятся и организуются данные, файлы, документы и другие ресурсы. Он предоставляет единый доступ к данным для разных пользователей и приложений и обеспечивает их сохранность, надежность и консистентность.

В хранилище могут храниться различные типы данных, такие как программные файлы, документы, изображения, видео, аудио, базы данных и другие. Хранилище может предоставлять разные возможности для организации данных, такие как версионирование, контроль доступа, индексирование и поиск, а также функции совместной работы и управления изменениями.

Хранилище может быть реализовано с использованием различных технологий, таких как файловые системы, базы данных, облачные хранилища и другие. Оно является важным элементом в информационных системах и обеспечивает эффективное управление и использование данных.

**31. Рабочая копия (Working Copy)** - это копия репозитория в системе контроля версий (Version Control System, VCS), которая доступна для работы над изменениями без влияния на основной репозиторий. Рабочая копия содержит все файлы и историю изменений, связанные с проектом. При внесении изменений в рабочую копию, можно добавлять, изменять или удалять файлы, а затем сохранять изменения в репозитории.

**32. Тестирование ПО (Software Testing)** - это процесс проверки и оценки программного обеспечения на соответствие заданным требованиям и оценке его качества. Основная цель тестирования ПО заключается в выявлении дефектов, ошибок, недоработок и проблем, которые могут повлиять на корректность, надежность и производительность программы.

**33. Понятие управления контроля версиями (СКВ)** относится к процессу управления изменениями в программном обеспечении или других документах. СКВ позволяет отслеживать изменения, вносимые в файлы и документы, и хранить предыдущие версии содержимого. Он также предоставляет возможности сравнивать изменения и восстанавливать предыдущие версии, если это необходимо.

Применение системы контроля версий:

- Улучшение сотрудничества: СКВ позволяет нескольким пользователям работать над одним и тем же файлом, сливать изменения и смотреть, кто и когда внес какие изменения.

- Отслеживание изменений: СКВ позволяет легко отслеживать изменения в файлах, включая внесение новых функций, исправление ошибок и устранение недочетов.

- Восстановление истории версий: СКВ сохраняет историю всех изменений, сделанных в файле, и позволяет легко восстановить предыдущие версии в случае необходимости.

- Резервное копирование: СКВ обеспечивает резервное копирование файлов и документов, чтобы в случае потери данных можно было восстановить предыдущую версию.

**34. Классификация систем контроля версий (СКВ)** может быть основана на различных критериях. Одна из возможных классификаций включает следующие типы СКВ:

- Локальные СКВ: Это системы, в которых каждый пользователь хранит свои изменения локально на своем компьютере. Примером локальных СКВ является RCS (Revision Control System).

- Централизованные СКВ: Это системы, в которых все изменения хранятся на центральном сервере. Пользователи получают актуальные версии файлов из центрального сервера и вносят изменения в локальные копии файлов. Примерами централизованных СКВ являются CVS (Concurrent Versions System) и Subversion (SVN).

- Распределенные СКВ: Это системы, в которых каждый пользователь имеет полную копию репозитория, в которой хранятся все версии файлов. Пользователи могут вносить изменения и обмениваться ими между собой. Примерами распределенных СКВ являются Git и Mercurial.

Кроме того, можно также классифицировать СКВ по типу используемого хранилища (например, файловая система или база данных), типу доступа (только для чтения или с возможностью записи) и другим критериям.

**35. Контроль версий типа SVN (Subversion)** является одним из инструментов управления изменениями в программном обеспечении. Он позволяет разработчикам отслеживать и управлять изменениями в исходных кодах проектов, а также координировать работу над различными версиями программы.

Смысл контроля версий состоит в следующем:

1. Отслеживание изменений: SVN позволяет хранить исходные коды и все изменения, вносимые в них. Разработчики могут видеть, кому принадлежит изменение, что было изменено и когда происходило изменение. Это позволяет легко восстановить предыдущие версии кода и отслеживать историю изменений.

2. Координация работы: при работе в команде SVN обеспечивает синхронизацию изменений между различными разработчиками. Каждый разработчик может работать над своей версией кода, а затем объединить свои изменения с общим репозиторием. Это позволяет избежать конфликтов и упрощает совместную разработку.

3. Управление версиями: SVN позволяет хранить несколько версий программы одновременно. Это позволяет вернуться к предыдущей версии, если новые изменения вызовут проблемы. Кроме того, SVN позволяет создавать теги и ветки, что упрощает управление выпусками и параллельное развитие различных версий программы.

Современные подходы к анализу ПО включают:

1. Статический анализ: это анализ исходного кода и других артефактов программы без ее выполнения. Он позволяет выявить потенциальные ошибки, уязвимости, неправильное использование API и другие проблемы.

2. Динамический анализ: это анализ программы во время ее выполнения. Он позволяет выявить проблемы производительности, утечки памяти, неправильную работу алгоритмов и другие проблемы, которые могут проявиться только при выполнении программы.

3. Анализ зависимостей: это анализ связей между различными компонентами программы. Он позволяет выявить слабые места, лишние зависимости и проблемы с модульностью.

4. Анализ данных: это анализ данных, используемых или создаваемых программой. Он позволяет выявить проблемы с правильностью данных, нарушениями целостности и другие проблемы, связанные с обработкой данных.

**36. Уровни тестирования включают:**

1. Модульное тестирование: тестирование отдельных модулей программы (функций, классов, методов) для проверки их корректности и работоспособности.

2. Интеграционное тестирование: тестирование взаимодействия между различными модулями программы для проверки их совместной работоспособности.

3. Системное тестирование: тестирование всей системы или приложения в целом для проверки его соответствия требованиям и поведению в реальных условиях использования.

4. Выходное: завершающий этап тестирования, на котором проверяется готовность ПП к поставке заказчику. Данный вид тестирования проводит независимый тестировщик.

5. Приемочное тестирование: тестирование системы или приложения в соответствии с требованиями и ожиданиями заказчика для проверки его готовности к эксплуатации.

**37. Импорт и экспорт настроек: общие сведения, применение.**

Импорт и экспорт настроек - это процессы переноса настроек и конфигурационных данных между программами, приложениями или устройствами. Эти настройки могут включать параметры пользовательского интерфейса, предпочтения пользователей, настройки безопасности и другие релевантные данные.

Применение импорта и экспорта настроек может быть разнообразным. Некоторые из основных применений включают:

1. Перенос настроек на новое устройство: Пользователи могут экспортировать свои настройки с одного устройства и импортировать их на новый, чтобы сохранить предпочтения и параметры конфигурации.

2. Обмен данными между программами: Импорт и экспорт настроек позволяет программам передавать данные между собой, чтобы обеспечить совместимость и переносимость.

3. Создание резервной копии и восстановление настроек: Пользователи могут экспортировать настройки в файл или облачное хранилище для создания резервной копии, которую можно восстановить в случае сбоя или потери данных.

4. Массовое применение настроек: Некоторые программы позволяют экспортировать настройки в файл, который может быть быстро импортирован на другие устройства или учетные записи пользователей, чтобы упростить настройку.

В целом, импорт и экспорт настроек позволяют пользователям сохранять, передавать и восстанавливать свои конфигурационные данные, обеспечивая удобство и гибкость использования программ и устройств.

**38. Обратное проектирование: основные понятия, виды применяемых инструментов и описание процесса.**

Обратное проектирование (reverse engineering) - это процесс анализа уже существующего продукта или системы с целью извлечения информации о его структуре, функциональности и дизайне. Оно обычно применяется для восстановления документации, понимания работы системы или создания нового продукта, совместимого с исходным.

Виды применяемых инструментов обратного проектирования могут включать:

1. Дизассемблеры: Исследуют исполняемые файлы или код программы и восстанавливают исходный код, на котором была основана программа.

2. Декомпиляторы: Извлекают исходный код из скомпилированного бинарного файла или исполняемого файла.

3. Статический анализатор кода: Используется для анализа кода и выявления зависимостей, структуры и функциональности.

4. Программы для анализа данных исходного кода: Эти инструменты позволяют обрабатывать и анализировать исходный код для выявления структуры и функциональности.

**39. Применение и функции обратного проектирования:**

Обратное проектирование (Reverse Engineering) является процессом извлечения информации о техническом объекте из уже существующего продукта или устройства. Применение и функции обратного проектирования включают:

1. Разработка резервных копий и восстановление: Обратное проектирование может использоваться для создания резервных копий устаревших или дефектных устройств, а также для восстановления утерянной документации или конструкции.

2. Исследование конкурентных продуктов: Обратное проектирование может использоваться для изучения конкурентных продуктов с целью определения их особенностей, структуры и функций. Это позволяет разработчикам улучшить свои собственные продукты и найти новые решения.

3. Модификация и улучшение существующих продуктов: Обратное проектирование позволяет анализировать и модифицировать существующие продукты, добавлять новые функции или улучшать их производительность и эффективность.

4. Интеграция систем и процессов: Обратное проектирование позволяет сопоставить различные системы и процессы для обеспечения их совместной работы и соответствия требованиям.

5. Улучшение безопасности и снижение рисков: Обратное проектирование помогает выявлять уязвимости и риски существующих систем и устройств, что позволяет разработать меры для улучшения безопасности и снижения возможных проблем.

**40. Описание анализа обратного проектирования:**

Анализ обратного проектирования включает в себя следующие этапы:

1. Сбор информации: На этом этапе проводится сбор исходных данных о продукте или устройстве, включая его физические свойства, электрические схемы, программное обеспечение и другую важную информацию.

2. Документация: Собранная информация о продукте документируется, чтобы обеспечить хранение и организацию данных для дальнейшего анализа и использования.

3. Обратный инжиниринг: На этом этапе проводится разбор продукта с целью выявления его структуры, функций и принципов работы. Для этого могут использоваться инструменты, такие как детальное изучение схем, разборка устройства или анализ программного кода.

4. Анализ данных: Полученная информация изучается и анализируется с целью выявления основных особенностей и характеристик продукта, его компонентов и процессов.

5. Восстановление и моделирование: На основе полученных данных и анализа происходит восстановление структуры и функций продукта. Это может включать создание 3D-модели, структурных диаграмм или программного кода.

6. Оценка и улучшение: Полученные результаты анализа используются для оценки существующего продукта и возможности его улучшения. Это может включать рекомендации по изменению дизайна, добавлению новых функций или улучшению качества и производительности.

Анализ обратного проектирования позволяет лучше понять существующие продукты и устройства, а также использовать эту информацию для разработки новых и улучшенных решений.

**41. Модульное тестирование (unit testing)** - процесс проверки отдельных программных процедур и подпрограмм, входящих в состав программ или программных систем.

Модульное тестирование производится непосредственным разработчиком и позволяет проверять все внутренние структуры и потоки данных в каждом модуле. Этот вид тестирования является частью этапа разработки. При модульном тестировании выполняется набор тестов, определяемый разработчиком так, чтобы охват тестированием каждого модуля был не менее 70-75%.

Элементами модульного тестирование являются:

* синтаксическая проверка — проверка с использованием некоторого инструментального средства для выявления синтаксических ошибок в программном коде;
* проверка соответствия стандартам кодирования — проверка кода на соответствие стандартам кодирования компании;
* технический обзор программного кода.

После успешного завершения модульного тестирования все измененные модули и наборы тестов сохраняются в БД проекта.

**42. Интеграционное тестирование (integration testing)** - Оно осуществляется независимым тестировщиком и входит в состав этапа тестирования.

Элементами интеграционного тестирования являются:

* проверка функциональности — проверка соответствия отдельных функций, выполняемых совокупностями модулей, функциям, заданным в спецификациях требований
* проверка промежуточных результатов — проверка всех промежуточных результатов и файлов на наличие и корректность
* проверка интеграции — проверка того, что модули передают друг другу информацию корректно

Ошибки, выявленные в ходе интеграционного тестирования, заносятся в БД ошибок.

Результаты интеграционного тестирования включаются в отчет о ходе тестирования при завершении цикла тестирования.

**43. Системное тестирование: основные понятия, элементы, результат.**

Системное тестирование - предназначен для проверки программной системы в целом, ее организации и функционирования на соответствие спецификациям требований заказчика.

Его проводит независимый тестировщик после успешного завершения интеграционного тестирования.

Элементами системного тестирования являются:

* граничное тестирование — тестирование в граничных условиях;
* прогоночное тестирование — тестирование всех функциональных характеристик реальной работы системы;
* целевое тестирование — тестирование на целевой платформе (по возможности)
* проверка документации — проверка пользовательской документации на корректность)
* другие тесты, определяемые тестировщиком

Ошибки, выявленные при системном тестировании, заносятся в БД проекта.

Результаты системного тестирования включаются в отчет о ходе тестирования.

**44. Локальный репозиторий: определение и функции.**

Локальный репозиторий - это хранилище кода и версий программы на компьютере разработчика. Он содержит все файлы, необходимые для работы с системой управления версиями (например, Git), и предоставляет возможность отслеживать изменения в коде, создавать и переключаться между разными ветками разработки, а также вносить изменения и сохранять их историю.

Функции локального репозитория:

1. Хранение версий программы: локальный репозиторий сохраняет все изменения в коде, включая создание, изменение и удаление файлов, чтобы разработчик мог отслеживать историю изменений.

2. Ветвление и слияние веток: разработчик может создавать разные ветки разработки, чтобы работать над разными функциями или исправлениями ошибок. Локальный репозиторий позволяет объединять изменения из разных веток или переключаться между ними.

3. Откат к предыдущим версиям: локальный репозиторий позволяет разработчику откатываться к предыдущим версиям программы, что полезно при исправлении ошибок или восстановлении работоспособности программы.

4. Работа в автономном режиме: локальный репозиторий позволяет разработчику работать с кодом без подключения к удаленному репозиторию. Это полезно, когда разработчик находится вне сети или не имеет доступа к удаленному репозиторию.

**45. Тестирование методом «серый ящик»** - это подход к тестированию программного обеспечения, который основан на знании некоторой информации о структуре и внутреннем устройстве тестируемой системы. В этом методе тестирования используется комбинация знаний о внутренней структуре системы и требованиях к ее функциональности. Тестирование «серым ящиком» позволяет тестировщику иметь представление о внутренней структуре системы и использовать эту информацию для разработки эффективных тестовых сценариев.

Процесс тестирования методом «серый ящик» обычно включает следующие шаги:

1. Анализ требований: изучение требований к системе и определение, какие аспекты внутренней структуры системы следует учесть при планировании и разработке тестовых случаев.

2. Изучение дизайна: анализ структуры и архитектуры системы, чтобы понять, как различные компоненты взаимодействуют, и определить, какие тестовые сценарии можно разработать.

3. Разработка тестовых случаев: на основе анализа требований и дизайна системы разработка тестовых сценариев для оценки функциональности и надежности системы.

4. Исполнение тестовых случаев: запуск разработанных тестовых сценариев и регистрация результатов тестирования.

5. Анализ результатов: анализ результатов тестирования, выявление дефектов и взаимодействие с разработчиками для исправления проблем.

**46. Получение доступа к GitHub** - это процесс создания аккаунта и получения доступа к платформе GitHub, где хранятся репозитории с открытым и закрытым кодом. Основные функции GitHub включают:

- Хранение кода: GitHub предоставляет возможность хранить и управлять версиями кода в репозиториях. Это позволяет командам разработчиков совместно работать над проектами и отслеживать изменения в коде.

- Контроль версий: GitHub основан на системе контроля версий Git, что позволяет разработчикам отслеживать изменения в коде, внести исправления и отменить изменения при необходимости.

- Коллаборация: разработчики могут совместно работать над проектами, создавая форки и подтверждая изменения через механизмы запросов на слияние (Pull Requests).

- Отслеживание ошибок: GitHub предоставляет возможность создания задач и отслеживания ошибок через встроенную систему отслеживания ошибок (Issue Tracking).

- Документация: разработчики могут создавать и поддерживать документацию для своих проектов на GitHub с помощью инструментов вроде Markdown и GitHub Pages.

Принцип GitHub основывается на открытой и коллаборативной разработке, предоставляя разработчикам возможность совместной работы над проектами и обмена кодом и идеями с другими участниками сообщества разработчиков.

**47. Организация разработки в сервисе GitHub:**

Организация разработки в сервисе GitHub может быть выполнена через использование функций и инструментов, предоставляемых платформой. Некоторые основные компоненты организации разработки в GitHub включают в себя:

- Репозитории: Репозитории являются центральными элементами разработки в GitHub. Они представляют собой места для хранения и управления кодом, документацией и другими файлами проекта.

- Коллабораторы: Коллабораторы - это люди, которым разрешено вносить изменения в репозитории. Они могут форкать репозитории, создавать ветки, делать коммиты и создавать запросы на слияние.

- Ветвление и слияние: GitHub предоставляет возможность создания веток для разработки новых функций или исправлений ошибок на основе существующего кода в репозитории. После завершения работы над веткой ее можно включить обратно в основную ветку с помощью запроса на слияние.

- Запросы на слияние: Запросы на слияние (Pull requests) предоставляют механизм для предложения изменений в коде репозитория и обсуждения с другими разработчиками. Они позволяют улучшить и контролировать качество кода перед его интеграцией в основную ветку.

- Проблемы и задачи: GitHub включает раздел для отслеживания проблем, задач и их обсуждения. Разработчики могут создавать задачи, указывать ответственных, обсуждать детали и контролировать прогресс.

- Обзор кода: GitHub предоставляет функциональность для обзора кода, позволяя другим разработчикам комментировать и оценивать предложенные изменения перед слиянием.

**48. Описание уровней доступа GitHub:**

В GitHub существуют различные уровни доступа, которые определяют, какие действия и функции доступны пользователям в рамках репозитория или организации. Некоторые общие уровни доступа в GitHub включают в себя:

- Владелец: Владелец - это уровень доступа, который предоставляет полный контроль над репозиторием или организацией. Владелец может управлять доступом и разрешениями других пользователей, создавать и удалять ветки, управлять запросами на слияние и другими аспектами разработки проекта.

- Администратор: Такой уровень доступа предоставляет пользователю возможность управлять настройками репозитория. Администратор может управлять коллабораторами, создавать и удалять ветки, отслеживать проблемы, настраивать предупреждения и т.д.

- Коллаборатор: Коллаборатор имеет доступ для внесения изменений в репозиторий. Он может создавать ветки, делать коммиты и создавать запросы на слияние.

- Член организации: Уровень доступа для пользователя, являющегося членом организации, определяет его роли и привилегии в контексте данной организации. Член организации может иметь различные уровни доступа в зависимости от настроек организации.

- Публичный доступ: Некоторые репозитории могут быть публичными, что означает, что любой пользователь может просматривать код, создавать запросы на слияние и вносить комментарии, но не может делать прямые изменения.

**49. Характеристика способов слияния копий репозитория.**

Существует несколько способов слияния копий репозитория. Вот некоторые из них:

- Слияние слиянием (Merge): Этот способ объединяет изменения из одной ветки (или веток) в другую. При слиянии Git автоматически пытается объединить изменения, если они не конфликтуют. Если возникают конфликты, Git позволяет разработчику решить их вручную. Слияние слиянием часто используется при работе с командой, где каждый член команды делает изменения в отдельной ветке и затем сливает их в основную ветку.

- Перебазирование (Rebase): В этом способе изменения из одной ветки ветки переносится в другую. В отличие от слияния слиянием, при перебазировании история коммитов может измениться, поскольку коммиты применяются поверх другой ветки. Это полезно, если вы хотите добавить свои изменения поверх актуальной версии кода. Однако перебазирование может создать сложности, если другие разработчики уже базируют свои изменения на вашей ветке.

- Вилка (Fork): Этот способ представляет собой создание копии репозитория на GitHub или другой платформе с хостингом Git. Вилка позволяет другим разработчикам вносить изменения в код, не влияя на оригинальный репозиторий. При необходимости владельцы оригинального репозитория могут интегрировать изменения из вилки.

**50. Модели создания кода в репозитории: описание и графическое представление.**

В репозитории можно использовать различные модели создания кода. Вот некоторые из них с их описанием и графическим представлением:

- Централизованная модель: В этой модели есть единственный центральный репозиторий, куда все разработчики вносят свои изменения. Один разработчик может захватить блокировку на изменение кода, что приводит к дублированию работы и замедляет процесс разработки.

Графическое представление:

```

┌─────────┐ pushes to ┌─────────┐

│ │ ─ ─ ─ ─ ─>│ │

│ Central │ │ Remote │

│ Repo │ < ─ ─ ─ ─ ┤ Repo │

│ │ < ─ ─ ─ ─ ┤ │

└─────────┘ pulls └─────────┘

```

- Распределенная модель: В этой модели каждый разработчик имеет свою копию репозитория. Они могут работать независимо, делая изменения и выкладывая их в центральный репозиторий или распространяя их между собой. Это позволяет разработчикам работать над разными частями проекта одновременно и ускоряет процесс разработки.

Графическое представление:

```

┌ ─ ─ ─ ─ ─ ─ ─ ┐ push ┌ ─ ─ ─ ─ ─ ─ ─ ┐

│ Developer 1 │ ─ ─ ─ ─ ─>│ Central │

│ Repo │ │ Repo │

│ │ < ─ ─ ─ ─ ┤ │

└ ─ ─ ─ ─ ─ ─ ─ ┘ pull └ ─ ─ ─ ─ ─ ─ ─ ┘

▲

│

│ pull

│

┌ ─ ─ ─ ─ ─ ─ ─ ┐ push ┌ ─ ─ ─ ─ ─ ─ ─ ┐

│ Developer 2 │ ─ ─ ─ ─ ─>│ Central │

│ Repo │ │ Repo │

│ │ < ─ ─ ─ ─ ┤ │

└ ─ ─ ─ ─ ─ ─ ─ ┘ pull └ ─ ─ ─ ─ ─ ─ ─ ┘

```

- Gitflow: Это модель, представляющая собой стратегию организации веток в репозитории. Она определяет основные ветки, такие как "master" (основная версия продукта) и "develop" (разрабатываемая версия продукта), а также вспомогательные ветки для функциональных и релизных версий. Эта модель обеспечивает хорошую организацию работы над разными функциональностями и выпускает стабильные релизы.

Графическое представление:

```

┌─────────────┐ ┌─────────────────┐

│ Release │ <───────│ Develop │

│ Branch │ └─────────────────┘

│ │

└─────────────┘

▲

│

┌───────┴───────┐

│ │

┌────────┴──────┐ ┌─────┴─────────┐

│ Feature │ │ Hotfix │

│ Branch │ │ Branch │

└────────┬──────┘ └─────┬─────────┘

│ │

**51. Архитектура MVC (Model-View-Controller)** – это популярный подход к организации структуры и разработке программного обеспечения. Основная идея заключается в разделении приложения на три основные составляющие:

- Модель (Model): отвечает за обработку данных, бизнес-логику и хранение состояния приложения. Модель не зависит от других компонентов архитектуры и не отображает данные напрямую.

- Представление (View): отвечает за отображение информации пользователю. Она получает данные от модели и представляет их в нужной форме, например, в виде HTML-страницы. Представление не содержит логики и не модифицирует данные.

- Контроллер (Controller): обрабатывает пользовательский ввод, взаимодействует с моделью и выбирает подходящее представление для отображения данных пользователю. Контроллер принимает решения на основе пользовательского ввода и взаимодействует с моделью, чтобы получить и обновить данные.

Общая структура MVC-архитектуры может быть представлена следующим образом:

+------------+

| Model |

+------------+

↑

| Получает данные

|

+-----+-----+-----+

| View |

+-----+-----+-----+

↑

| Отображает данные

|

+------------+-----+

| Controller |

+------------+-----+

↑

| Взаимодействует с моделью и представлением

**52. Процесс реализации проверки на стороне клиента (client-side validation) обычно включает следующие шаги:**

1. Добавление формы: Необходимо создать HTML-форму, содержащую поля, которые должны быть проверены на стороне клиента. Каждому полю формы можно присвоить атрибуты, такие как "required" или "pattern", чтобы указать необходимость заполнения поля или задать шаблон проверки.

2. Написание JavaScript-кода: С использованием JavaScript можно добавить проверку на стороне клиента для формы. Например, можно использовать объекты Document и Element для доступа к элементам формы, а затем провести проверку значений полей формы на основе определенных правил и условий.

3. Обработка событий: Чтобы активировать проверку на стороне клиента, нужно добавить обработчики событий к форме или отдельным элементам формы. Например, можно добавить событие "submit" к форме, чтобы проверить все поля перед отправкой данных на сервер.

4. Отображение сообщений об ошибке: Если проверка на стороне клиента обнаруживает ошибки, необходимо отобразить соответствующие сообщения об ошибках пользователю. Можно использовать элементы HTML, такие как <span> или <div>, чтобы отобразить сообщения рядом с соответствующими полями формы.

5. Комбинирование с серверной проверкой: Проверка на стороне клиента не должна заменять серверную проверку. Для обеспечения безопасности и целостности данных следует провести также проверку на стороне сервера.

В результате, процесс реализации проверки на стороне клиента позволяет проверить введенные пользователем данные до отправки их на сервер, что помогает повысить удобство использования и уменьшить нагрузку на сервер.

**53. JavaScript: инструменты языка. Состав среды разработки (IDE).**

JavaScript имеет широкий набор инструментов, которые помогают разработчикам в создании и отладке кода на этом языке. Некоторые из наиболее популярных инструментов включают в себя:

1. Редакторы кода: Редакторы кода, такие как Visual Studio Code, Sublime Text и Atom, обеспечивают функции подсветки синтаксиса, автодополнения кода, настраиваемые настройки форматирования и другие удобные функции для работы с JavaScript.

2. Отладчики: Браузеры, такие как Google Chrome и Firefox, предоставляют интегрированные инструменты для отладки JavaScript. Эти инструменты позволяют разработчикам просматривать и изменять значения переменных, отслеживать выполнение кода, а также находить и исправлять ошибки.

3. Пакетные менеджеры: JavaScript имеет несколько пакетных менеджеров, таких как npm (Node Package Manager) и yarn, которые позволяют разработчикам управлять зависимостями проекта и устанавливать сторонние библиотеки и фреймворки.

4. Компиляторы и сборщики: Некоторые разработчики JavaScript используют компиляторы и сборщики, такие как Babel и Webpack, чтобы преобразовывать код на JavaScript в совместимую версию, объединять и оптимизировать файлы и дополнительно обрабатывать код.

**54. Описание процесса реализации проверки на стороне клиента** Преимущества проверки на сервере.

Проверка на стороне клиента - это процесс валидации информации, который выполняется на компьютере или устройстве пользователя, перед отправкой данных на сервер. Принцип работы проверки на стороне клиента может варьироваться в зависимости от используемых инструментов и технологий, но обычно он включает следующие шаги:

1. Сбор данных: Данные, введенные пользователем в форму или другой интерфейс, собираются и сохраняются в переменных или объектах JavaScript.

2. Валидация данных: С помощью JavaScript разработчик может проверить, соответствуют ли введенные данные определенным правилам или ограничениям. Это может включать проверку формата данных (например, электронной почты), длину или диапазон значений, требования к паролю и другие параметры.

3. Отображение ошибок: Если данные не соответствуют заданным правилам, JavaScript может создать и отобразить сообщения об ошибках рядом с неправильными полями или в другом удобном месте на странице.

Преимущества проверки на стороне клиента включают:

1. Быстрая отзывчивость: Проверка на стороне клиента позволяет пользователям получать обратную связь по мере ввода данных без необходимости отправлять данные на сервер для проверки. Это может улучшить пользовательский опыт и ускорить процесс.

2. Оптимизация нагрузки на сервер: Предварительная проверка на стороне клиента помогает фильтровать некорректные данные еще до их отправки на сервер. Это позволяет снизить нагрузку на сервер и уменьшить объем обработки неправильных данных на стороне сервера.

3. Уменьшение ошибок: Проверка на стороне клиента позволяет предотвращать отправку неправильных данных на сервер и уменьшает вероятность возникновения ошибок при обработке данных на сервере.

Однако важно отметить, что проверка на стороне клиента может быть обходной или игнорироваться злоумышленниками, поэтому всегда следует выполнять также проверку данных на стороне сервера для обеспечения безопасности и целостности системы.

**55. Библиотека классов** - это набор готовых классов и методов, которые можно использовать для разработки приложений. Она предоставляет возможность повторного использования кода, упрощает разработку и улучшает поддерживаемость программного обеспечения.

Доступ к членам класса в библиотеке классов может быть разным. В общем случае, доступ к членам класса контролируется модификаторами доступа, такими как public, protected, private. Public позволяет получить доступ к членам класса из любого места кода, protected - только из класса и его потомков, private - только из самого класса.

**56. Язык Ассемблера** - это низкоуровневый язык программирования, который предоставляет прямое управление аппаратурой компьютера. Он использует мнемонические коды для представления команд процессора и адресов памяти.

Основная возможность языка Ассемблера - это полный контроль над аппаратурой компьютера, что позволяет максимально оптимизировать код и достичь высокой производительности. Кроме того, язык Ассемблера обеспечивает прямое взаимодействие с регистрами процессора и памятью.

**57. Сегментная адресация** - это система адресации в компьютерных системах, которая позволяет адресовать данные, используя комбинацию сегментного и смещенного адресов. В данной системе адресация происходит не только по одному целому числу, но и по сегменту, что позволяет увеличить размер доступной памяти и легче организовать работу с данными.

Примеры применения сегментной адресации:

- Виртуальная память: сегментация позволяет эффективно использовать ограниченные ресурсы физической памяти, разбивая ее на сегменты различных размеров и загружая в память только те сегменты, которые нужны в данный момент.

- Защита памяти: сегментация позволяет выделить отдельные сегменты памяти для различных процессов или пользователей, обеспечивая защиту от несанкционированного доступа.

- Исполнение кода: сегментация может использоваться для хранения и выполнения кода программы, где каждый сегмент содержит определенный тип инструкций или данных.

**58. Дизассемблер** - это программа, которая преобразует машинный код программы в читаемый человеком вид. Он позволяет анализировать и понимать, как работает программа, и может быть полезен в отладке, обратной разработке или изучении программного обеспечения.

Возможности дизассемблера:

- Преобразование машинного кода в ассемблерный код: дизассемблер может преобразовывать машинный код в код на языке ассемблера, что позволяет легко прочитать и понять код программы.

- Анализ исполняемых файлов: дизассемблер может анализировать исполняемые файлы и извлекать информацию о функциях, переменных, константах и других элементах программы.

- Отладка: дизассемблер может помочь в отладке программы, позволяя разработчику анализировать состояние памяти, значения регистров и последовательность выполнения инструкций.

**59. Мониторинг работоспособности системы контроля версий: назначение и инструменты мониторинга.**

Мониторинг работоспособности системы контроля версий (СКВ) имеет критическое значение для обеспечения эффективной и надежной разработки программного обеспечения. Назначение мониторинга состоит в том, чтобы непрерывно отслеживать производительность и стабильность СКВ, обнаруживать и устранять проблемы, связанные с доступностью, нагрузкой и безопасностью.

Для мониторинга работоспособности системы контроля версий могут использоваться различные инструменты. Некоторые из них включают:

1. Метрики производительности: мониторинг производительности СКВ, такой как время выполнения операций, использование памяти и процессора, может помочь выявить узкие места и оптимизировать работу системы.

2. Мониторинг доступности: система мониторинга может проверять доступность СКВ, осуществляя пинги или отправляя запросы на сервер. Это позволяет своевременно обнаруживать проблемы с доступностью и предотвращать длительные простои.

3. Мониторинг безопасности: следует отслеживать любые потенциальные уязвимости или атаки на СКВ, используя механизмы безопасности, такие как регистрация всех действий пользователей и анализ журналов для обнаружения подозрительной активности.

4. Уведомления и оповещения: система мониторинга должна быть настроена на отправку уведомлений и оповещений в случае обнаружения проблем или аварийных ситуаций, чтобы разработчики могли оперативно реагировать и устранять проблемы.

**60. Коллективная разработка: характеристика, структура.**

Коллективная разработка - это методология разработки программного обеспечения, в которой несколько разработчиков совместно работают над одним проектом, используя систему контроля версий и совместный доступ к репозиторию. Целью коллективной разработки является повышение эффективности и качества разработки путем распределения задач и интеграции вклада различных участников.

Структура коллективной разработки может варьироваться в зависимости от размера команды и конкретного проекта, но обычно включает в себя следующие элементы:

1. Репозиторий: центральное место, где хранится код проекта и с которым все участники могут синхронизироваться и вносить свои изменения.

2. Ветвление и слияние: разработчики создают различные ветки для работы над своими задачами, после чего вносят свои изменения обратно в главную ветку проекта путем слияния.

3. Код-ревью: участники команды проверяют код других разработчиков, обсуждают его и предлагают улучшения или исправления.

4. Итеративность: разработка основывается на коротких итерациях, где задачи и прогресс регулярно обновляются и обсуждаются всей командой.

5. Совместное общение: команда использует коммуникационные инструменты, такие как чаты, форумы и системы отслеживания ошибок, для общения и обмена информацией.

Коллективная разработка позволяет быстрее достигать результата, повышает качество и позволяет распределить задачи в соответствии с компетенциями участников команды.

**61. Проверка сервера: характеристика, принцип работы, особенности.**

Проверка сервера – это процесс, когда осуществляется проверка работоспособности и доступности сервера. Это важная задача для обеспечения стабильной работы сети и предотвращения возможных проблем или сбоев.

Характеристики проверки сервера могут включать следующие аспекты:

- Проверка соединения: проверяется, доступен ли сервер и может ли клиент успешно установить связь с ним.

- Проверка производительности: измерение нагрузки на сервер, скорости передачи данных и времени отклика.

- Проверка безопасности: анализ уязвимостей, обнаружение атак и проверка конфигурации сервера с целью защиты от несанкционированного доступа и вредоносного программного обеспечения.

- Проверка доступности сервисов: проверка работоспособности различных сервисов на сервере, таких как веб-сайты, электронная почта, базы данных и другие.

Принцип работы проверки сервера обычно основан на отправке запросов от клиента к серверу и анализе полученных ответов. Для проверки доступности сервера используется различные протоколы, такие как ICMP (Internet Control Message Protocol) для проверки наличия соединения, HTTP для проверки веб-сервера и другие.

Особенности проверки сервера:

- Регулярность: проверка сервера рекомендуется проводить в регулярном режиме, чтобы обнаружить проблемы и аномалии как можно раньше.

- Автоматизация: проверка сервера может быть автоматизирована с использованием специальных программных инструментов или сервисов, что упрощает процесс и позволяет получить быстрый результат.

- Мониторинг: результаты проверки сервера могут быть интегрированы в системы мониторинга, чтобы оперативно уведомлять администраторов о возможных проблемах или отклонениях от нормы.

- Логирование: результаты проверки сервера обычно записываются в журнал для дальнейшего анализа или отслеживания изменений со временем.

**62. Проверка клиента, характеристика, принцип работы**.

Проверка клиента – это процесс, при котором осуществляется проверка работоспособности и настройки клиентского устройства, подключенного к сети, для обеспечения надежной связи и устранения возможных проблем.

Характеристики проверки клиента могут включать следующие аспекты:

- Проверка подключения: проверка наличия и стабильности подключения клиента к сети.

- Проверка настроек сети: анализ и проверка настроек IP-адреса, DNS, шлюза по умолчанию и других параметров.

- Проверка пропускной способности: измерение скорости передачи данных от клиента до сервера и обратно.

- Проверка безопасности: проверка наличия антивирусного ПО, обновленных программ и настроек безопасности для защиты клиента от вредоносных атак или утечек данных.

Принцип работы проверки клиента часто основывается на отправке запросов от клиента к серверу и анализе полученных ответов. Для проверки доступности клиента можно использовать различные методы, такие как пинг (отправка ICMP-запросов), HTTP-запросы и другие протоколы в зависимости от цели проверки.

Однако следует отметить, что проверка клиента может иметь другие особенности в зависимости от конкретной сетевой среды или типа проверяемых устройств. Например, в безопасных сетях могут использоваться специальные протоколы или сертификаты для проверки легитимности клиента перед разрешением доступа.

**63. Отличительные особенности IDE (интегрированной среды разработки) C++/C#:**

1. Встроенная отладка: IDE C++/C# обеспечивает мощный отладчик, позволяющий запускать код по шагам, подсматривать значения переменных и исправлять ошибки.

2. Авто дополнение и подсказки: IDE обладает функцией авто дополнения, которая предлагает варианты кода на основе того, что вы уже напечатали. Также она предоставляет подсказки по доступным методам и свойствам.

3. Интеграция с системами контроля версий: IDE поддерживает интеграцию с такими системами, как Git, что упрощает работу с кодом в команде.

4. Удобная навигация по коду: IDE C++/C# предлагает функции навигации по коду, позволяющие быстро переходить к определенным классам, методам или свойствам.

5. Интегрированная справка: Идеальная среда разработки обычно предлагает встроенную справку, которая содержит информацию о синтаксисе языка, доступных классах и методах, а также примеры использования.

**64. Отличительные особенности IDE Java:**

1. Компиляция в реальном времени: IDE Java сканирует ваш код на наличие ошибок и предупреждений во время его написания, что позволяет быстро обнаруживать проблемы и устранять их.

2. Мощный отладчик: IDE Java предоставляет отладчик с широкими функциональными возможностями, такими как шаг за шагом выполнение кода, просмотр значений переменных и анализ стека вызовов.

3. Поддержка статического анализа кода: Некоторые IDE Java имеют встроенные инструменты статического анализа кода, способные обнаружить потенциальные проблемы или несоответствия стандартам кодирования.

4. Интеграция с серверами приложений: IDE Java обычно поддерживает интеграцию со сторонними серверами приложений, что упрощает разработку, тестирование и развертывание Java-приложений.

5. Поддержка фреймворков и библиотек: IDE Java позволяет легко интегрировать и использовать различные фреймворки и библиотеки Java, обеспечивая улучшенную производительность разработки.

**65. Отключение JavaScript на стороне клиента: особенности.**

Отключение JavaScript на стороне клиента - это процесс отключения выполнения JavaScript-кода на веб-сайте веб-браузером пользователя. Возможны несколько основных особенностей этого процесса:

1. Отключение функциональности: JavaScript является мощным языком программирования, который используется для добавления интерактивности и динамических функций на веб-сайты. При его отключении на стороне клиента, функции, зависящие от JavaScript, могут быть отключены или работать неправильно.

2. Отключение валидации данных: JavaScript часто используется для валидации данных на веб-сайтах, чтобы предотвратить отправку неправильных данных на сервер. Если JavaScript отключен, то эта валидация не будет выполняться, что может привести к проблемам с неправильными данными на сервере.

3. Отключение AJAX-запросов: JavaScript является ключевым инструментом для выполнения асинхронных запросов на сервер с использованием технологии AJAX. При отключении JavaScript возможность отправлять и обрабатывать AJAX-запросы будет потеряна.

4. Отключение аналитики и отслеживания: Многие веб-сайты используют JavaScript для сбора аналитических данных и отслеживания пользователей. Если JavaScript отключен, эта функциональность также будет недоступна.

5. Попытка отключить уязвимости: JavaScript иногда может быть использован для проведения атак на веб-сайты, таких как внедрение вредоносного кода или кросс-сайтовый сценарий (XSS). Отключение JavaScript может снизить уязвимости веб-сайта к таким атакам.

**66. Команды github: git init, git branch, git push/pull характеристика, основные действия.**

- `git init` - команда используется для создания нового репозитория Git. После выполнения этой команды, в текущей директории будет создан пустой репозиторий Git, который будет использоваться для отслеживания изменений в файлах.

- `git branch` - команда используется для создания, переименования или удаления веток в Git. Она также позволяет просматривать список существующих веток.

- `git push` - команда используется для отправки ("пуша") локальных изменений в удаленный репозиторий Git. Это позволяет вам обновить удаленный репозиторий с вашими последними изменениями.

- `git pull` - команда используется для получения ("пулла") обновлений из удаленного репозитория Git и объединения их с вашим локальным репозиторием. Этот процесс позволяет вам получить последние изменения, сделанные другими разработчиками, и синхронизировать свой репозиторий с удаленным.

Основные действия, связанные с этими командами, включают создание репозитория, создание и переключение веток, добавление файлов в индекс, фиксацию изменений, отправку локальных изменений в удаленный репозиторий (push), получение обновлений из удаленного репозитория и их объединение с локальным (pull).

**67. Приемочное тестирование** - это вид тестирования программного обеспечения, проводимый организацией, отвечающей за инсталляцию, сопровождение программной системы и обучение конечного пользователя.

Основные понятия приемочного тестирования:

1. План приемочного тестирования - документ, в котором описывается подход к проведению приемочного тестирования, определяются его цели, задачи, ожидаемые результаты, а также ресурсы и расписание для выполнения тестирования.

2. Функциональные требования - это спецификация функциональных возможностей программного продукта, которые должны быть проверены во время приемочного тестирования.

3. Вариант использования - это сценарий или последовательность действий, которую пользователь проделывает с программным продуктом, чтобы достичь определенной цели. Во время приемочного тестирования тестируется поведение программы при различных вариантах использования.

Результат приемочного тестирования - это отчет, который содержит информацию о выполненных тестовых сценариях, ошибках, обнаруженных в процессе тестирования, и общую информацию о соответствии программного продукта требованиям.

**68. Выходное тестирование** - завершающий этап тестирования на котором проверяется готовность ПП к поставке заказчику. Данный вид тестирования проводит независимый тестировщик.

Элементами выходного тестирования являются:

* проверка инсталляции — проверка на ясность и корректность инструкций по инсталляции;
* проверка документации — проверка того, что вся необходимая документация полностью подготовлена и готова к передаче заказчику.

Ошибки, выявленные при выходном тестировании, заносятся в БД проекта.

При успешном завершении выходного тестирования ПП поставляется заказчику вместе с отчетом о результатах тестирования. Результат выходного тестирования - это подтверждение корректной работы программы после ее модификации или исправления ошибок.

**69. Основные элементы для тестирования ПО и их результаты:**

1. Планирование тестирования: результат - план тестирования, который включает в себя описание тестовых сценариев, ресурсы и расписание.

2. Проектирование тестов: результат - тестовая документация, включающая тестовые случаи, условия и ожидаемые результаты.

3. Создание тестовых данных: результат - набор данных, необходимых для проведения тестов.

4. Выполнение тестов: результат - запись результатов тестов и обнаруженных дефектов.

5. Отчет о тестировании: результат - отчет, содержащий информацию о проведенных тестах, обнаруженных дефектах и рекомендациях по их исправлению.

**70. Атрибуты и назначение команд git rebase, git clone и git log**:

1. git rebase: команда git rebase используется для переоснования (rebase) изменений из одной ветки на другую. Основные атрибуты команды git rebase включают:

- -i (--interactive): позволяет выполнять интерактивное переоснование, включая возможность редактирования коммитов и их порядка.

- -m (--merge): указывает git на выполнение merge при переосновании, а не простого применения изменений.

2. git clone: команда git clone используется для создания копии удаленного репозитория на локальном компьютере. Основные атрибуты команды git clone включают:

- --branch <branch\_name>: позволяет склонировать только указанную ветку.

- --depth <depth>: указывает максимальную глубину истории, которую нужно скачать.

3. git log: команда git log используется для просмотра информации о коммитах в репозитории. Основные атрибуты команды git log включают:

- --oneline: выводит краткую информацию о каждом коммите на одной строке.

- --author <author\_name>: фильтрует коммиты только по указанному автору.

Применение данных команд зависит от конкретных потребностей разработчика и команды. Например, команда git rebase может использоваться для объединения истории коммитов в более логичный порядок, а команда git clone - для получения локальной копии удаленного репозитория для работы над проектом. Команда git log может быть использована для анализа истории коммитов и поиска конкретных изменений.

1. **Сравнить функциональные особенности калькулятора, представленного в ОС Windows и предустановленного в папке «экзамен» на рабочем столе под названием «1.exe» и описать отличительные черты каждого из них. Результат оформить в документе Word в виде таблицы.**

Сравнение проводить по следующим оценочным элементам:

- надежность ПС,

- сопровождаемость,

- корректность.

Критерии оценки (1 или 0)

Всесравнение занести в следующую таблицу

Результат представлен на таблице 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код элемента | Наименование | Метод оценки | Оценка калькулятора фирмы  Microsoft | Оценка калькулятора |
| Оценочные элементы фактора «Надежность ПС» | | | | |
| 1 | Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных | Экспертный | 1 | 1 |
| 2 | Возможность обработки ошибочных ситуаций | Экспертный | 1 | 1 |
| 3 | Полнота обработки ошибочных данных | Экспертный | 1 | 1 |
| 4 | Наличие тестов для проверки допустимых значений входных данных | Экспертный | 0 | 0 |
| 5 | Наличие системы контроля полноты входных данных | Экспертный | 1 | 1 |
| 6 | Наличие средств контроля корректности входных данных | Экспертный | 1 | 1 |
| 7 | Наличие требований к программе по восстановлению процесса выполнения в случае сбоя ОС, внешних устройств, процессора | Экспертный | 1 | 1 |
| 8 | Наличие требований к программе по восстановлению результатов при отказах ОС, внешних устройств, процессора | Экспертный | 0 | 0 |
| 9 | Наличие средств восстановления при сбоях оборудования | Экспертный | 1 | 1 |
| 10 | Наличие возможности повторного старта с точки прерывания | Экспертный | 1 | 0 |
| 11 | Наличие обработки неопределенностей | Экспертный | 1 | 1 |
| 12 | Наличие централизованного управления процессами, конкурирующими из-за ресурсов | Экспертный | 1 | 1 |
| 13 | Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления | Экспертный | 1 | 1 |
|  |  | Всего | 11 | 10 |
| Оценочные элементы фактора «Сопровождаемость» | | | | |
| 14 | Наличие комментариев в точках входа и выхода в программу | Экспертный | 1 | 0 |
| 15 | Осуществляется ли передача результатов работы модуля через вызывающий его модуль | Экспертный | 1 | 0 |
| 16 | Оценка программы по числу циклов | Экспертный | 1 | 1 |
| 17 | Используется ли язык высокого уровня | Экспертный | 1 | 1 |
| 18 | Наличие проверки корректности передаваемых данных | Экспертный | 1 | 1 |
| 19 | Использовании при построении программ метода структурного программирования | Экспертный | 1 | 1 |
| 20 | Соблюдение принципа процесса разработки программы сверху вниз | Экспертный | 1 | 1 |
| 21 | Наличие ограничений на размеры модуля | Экспертный | 1 | 1 |
| 22 | Наличие модульной схемы программы | Экспертный | 1 | 0 |
| 23 |  | Всего | 9 | 6 |
| Оценочные элементы фактора «Корректность» | | | | |
| 24 | Наличие всех необходимых документов для понимания и использования ПС | Экспертный | 1 | 0 |
| 25 | Наличие описание схемы иерархии модулей программы | Экспертный | 1 | 0 |
| 26 | Наличие описаний основных функций | Экспертный | 1 | 0 |
| 27 | Наличие описаний частных функций | Экспертный | 1 | 0 |
| 28 | Наличие описания данных | Экспертный | 1 | 0 |
| 29 | Наличие описания алгоритмов | Экспертный | 1 | 0 |
| 30 | Наличие описания интерфейсов между модулями | Экспертный | 0 | 0 |
| 31 | Наличие описания всех параметров | Экспертный | 0 | 0 |
| 32 | Наличие описание методов настройки системы | Экспертный | 1 | 0 |
| 33 | Наличие описание способов проверки работоспособности программы | Экспертный | 0 | 0 |
| 34 | Реанимация всех модулей системы | Экспертный | 1 | 0 |
| 35 | Реанимация всех основных функций | Экспертный | 1 | 1 |
| 36 | Реанимация всех алгоритмов | Экспертный | 1 | 0 |
| 37 | Наличие определений всех данных: переменные, индексы., массивы и пр. | Экспертный | 1 | 0 |
| 38 | Наличие интерфейсов с пользователем | Экспертный | 1 | 1 |
| 39 | Отсутствие противоречий в выполнении основных функций | Экспертный | 1 | 0 |
| 40 | Отсутствие противоречий в выполнении частных функций | Экспертный | 1 | 1 |
| 41 | Отсутствие противоречий в выполнении алгоритмов | Экспертный | 1 | 1 |
| 42 | Правильность взаимосвязей | Экспертный | 1 | 1 |
| 43 | Правильность реализаций интерфейса с пользователем | Экспертный | 1 | 1 |
| 44 | Отсутствие противоречий в настройке системы | Экспертный | 1 | 1 |
| 45 | Комплектность документации соответствии со стандартами | Экспертный | 0 | 0 |
|  |  | Всего | 18 | 8 |

**Выводы:** научился проводить оценку качества программного средства по различным показателям.

Калькулятор фирмы Microsoft преодбладает большим приимуществом (38 баллов), потому что над этим программным продуктом работало подразделение-отдел крупной корпорации с предоставлением продукта на рынке более 37 лет(на 07.09.23) и множество ошибок, «багов и вылетов» исправлены со временем.

Написанный студентами (24 баллов) был создан в рамках обучения и не включает в себя подробные инструкции и модули/алгоритмы решения исключительных задач, приводящие к ошибке и закрытию программы.

2. На основании основных метрик описать функциональные особенности представленной в папке «экзамен» на рабочем столе программы под названием «2.exe». Определить основную цель приложения. Результат оформить в документе Word в виде таблицы.

Метрики описания функциональных особенностей представленной программы могут включать:

1. Общее описание программы: Основной целью программы является...

2. Описание возможностей программы: Программа позволяет пользователю...

3. Описание основных функций программы: Программа включает следующие функции...

4. Описание алгоритмов и методов, используемых в программе: Для достижения своих функций, программа использует следующие алгоритмы и методы...

5. Описание интерфейса пользователя: Программа имеет удобный и интуитивно понятный интерфейс, позволяющий пользователю...

6. Описание входных и выходных данных: При запуске программы пользователь может предоставить следующие входные данные..., а программа возвращает следующие выходные данные...

7. Описание возможных ошибок и исключительных ситуаций: В случае возникновения ошибки, программа сообщает пользователю следующую информацию и предлагает возможные решения...

8. Описание требований к системе и зависимостей: Для работы программы необходима следующая системная конфигурация и зависимости...

9. Описание процесса установки и настройки программы: Чтобы установить программу, пользователь должен выполнить следующие шаги...

10. Описание поддержки и обновлений: Программа также предлагает следующую поддержку и возможности обновления...

*Оценочные элементы фактора «Надежность ПС»*

Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных; Возможность обработки ошибочных ситуаций; Полнота обработки ошибочных данных; Наличие тестов для проверки допустимых значений входных данных; Наличие системы контроля полноты входных данных; Наличие средств контроля корректности входных данных; Наличие требований к программе по восстановлению процесса выполнения в случае сбоя ОС, внешних устройств, процессора; Наличие требований к программе по восстановлению результатов при отказах ОС, внешних устройств, процессора;

Наличие средств восстановления при сбоях оборудования; Наличие возможности повторного старта с точки прерывания Наличие обработки неопределенностей; Наличие централизованного управления процессами, конкурирующими из-за ресурсов; Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления;

*Оценочные элементы фактора «Сопровождаемость»*

Наличие комментариев в точках входа и выхода в программу; Осуществляется ли передача результатов работы модуля через вызывающий его модуль; Оценка программы по числу циклов;

Используется ли язык высокого уровня; Наличие проверки корректности передаваемых данных; Использовании при построении программ метода структурного программирования; Соблюдение принципа процесса разработки программы сверху вниз; Наличие ограничений на размеры модуля; Наличие модульной схемы программы.

*Оценочные элементы фактора «Корректность»*

Наличие всех необходимых документов для понимания и использования ПС; Наличие описание схемы иерархии модулей программы; Наличие описаний основных функций; Наличие описаний частных функций; Наличие описания данных; Наличие описания алгоритмов; Наличие описания интерфейсов между модулями; Наличие описания всех параметров; Наличие описание методов настройки системы; Наличие описание способов проверки работоспособности программы; Реанимация всех модулей системы; Реанимация всех основных функций; Реанимация всех алгоритмов; Наличие определений всех данных: переменные, индексы., массивы и пр. ; Наличие интерфейсов с пользователем; Отсутствие противоречий в выполнении основных функций; Отсутствие противоречий в выполнении частных функций; Отсутствие противоречий в выполнении алгоритмов; Правильность взаимосвязей; Правильность реализаций интерфейса с пользователем; Отсутствие противоречий в настройке системы;Комплектность документации соответствии со стандартами;

**3. На основании основных метрик описать функциональные особенности представленной в папке «экзамен» на рабочем столе программы под названием «3.exe». Определить основную цель приложения. Результат оформить в документе Word в виде таблицы.**

Метрики описания функциональных особенностей представленной программы могут включать:

1. Общее описание программы: Основной целью программы является...

2. Описание возможностей программы: Программа позволяет пользователю...

3. Описание основных функций программы: Программа включает следующие функции...

4. Описание алгоритмов и методов, используемых в программе: Для достижения своих функций, программа использует следующие алгоритмы и методы...

5. Описание интерфейса пользователя: Программа имеет удобный и интуитивно понятный интерфейс, позволяющий пользователю...

6. Описание входных и выходных данных: При запуске программы пользователь может предоставить следующие входные данные..., а программа возвращает следующие выходные данные...

7. Описание возможных ошибок и исключительных ситуаций: В случае возникновения ошибки, программа сообщает пользователю следующую информацию и предлагает возможные решения...

8. Описание требований к системе и зависимостей: Для работы программы необходима следующая системная конфигурация и зависимости...

9. Описание процесса установки и настройки программы: Чтобы установить программу, пользователь должен выполнить следующие шаги...

10. Описание поддержки и обновлений: Программа также предлагает следующую поддержку и возможности обновления...

*Оценочные элементы фактора «Надежность ПС»*

Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных; Возможность обработки ошибочных ситуаций; Полнота обработки ошибочных данных; Наличие тестов для проверки допустимых значений входных данных; Наличие системы контроля полноты входных данных; Наличие средств контроля корректности входных данных; Наличие требований к программе по восстановлению процесса выполнения в случае сбоя ОС, внешних устройств, процессора; Наличие требований к программе по восстановлению результатов при отказах ОС, внешних устройств, процессора;

Наличие средств восстановления при сбоях оборудования; Наличие возможности повторного старта с точки прерывания Наличие обработки неопределенностей; Наличие централизованного управления процессами, конкурирующими из-за ресурсов; Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления;

*Оценочные элементы фактора «Сопровождаемость»*

Наличие комментариев в точках входа и выхода в программу; Осуществляется ли передача результатов работы модуля через вызывающий его модуль; Оценка программы по числу циклов;

Используется ли язык высокого уровня; Наличие проверки корректности передаваемых данных; Использовании при построении программ метода структурного программирования; Соблюдение принципа процесса разработки программы сверху вниз; Наличие ограничений на размеры модуля; Наличие модульной схемы программы.

*Оценочные элементы фактора «Корректность»*

Наличие всех необходимых документов для понимания и использования ПС; Наличие описание схемы иерархии модулей программы; Наличие описаний основных функций; Наличие описаний частных функций; Наличие описания данных; Наличие описания алгоритмов; Наличие описания интерфейсов между модулями; Наличие описания всех параметров; Наличие описание методов настройки системы; Наличие описание способов проверки работоспособности программы; Реанимация всех модулей системы; Реанимация всех основных функций; Реанимация всех алгоритмов; Наличие определений всех данных: переменные, индексы., массивы и пр. ; Наличие интерфейсов с пользователем; Отсутствие противоречий в выполнении основных функций; Отсутствие противоречий в выполнении частных функций; Отсутствие противоречий в выполнении алгоритмов; Правильность взаимосвязей; Правильность реализаций интерфейса с пользователем; Отсутствие противоречий в настройке системы;Комплектность документации соответствии со стандартами;

**4. На основании основных метрик описать функциональные особенности представленной в папке «экзамен» на рабочем столе программы под названием «4.exe». Определить основную цель приложения. Результат оформить в документе Word в виде таблицы.**

Метрики описания функциональных особенностей представленной программы могут включать:

1. Общее описание программы: Основной целью программы является...

2. Описание возможностей программы: Программа позволяет пользователю...

3. Описание основных функций программы: Программа включает следующие функции...

4. Описание алгоритмов и методов, используемых в программе: Для достижения своих функций, программа использует следующие алгоритмы и методы...

5. Описание интерфейса пользователя: Программа имеет удобный и интуитивно понятный интерфейс, позволяющий пользователю...

6. Описание входных и выходных данных: При запуске программы пользователь может предоставить следующие входные данные..., а программа возвращает следующие выходные данные...

7. Описание возможных ошибок и исключительных ситуаций: В случае возникновения ошибки, программа сообщает пользователю следующую информацию и предлагает возможные решения...

8. Описание требований к системе и зависимостей: Для работы программы необходима следующая системная конфигурация и зависимости...

9. Описание процесса установки и настройки программы: Чтобы установить программу, пользователь должен выполнить следующие шаги...

10. Описание поддержки и обновлений: Программа также предлагает следующую поддержку и возможности обновления...

*Оценочные элементы фактора «Надежность ПС»*

Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных; Возможность обработки ошибочных ситуаций; Полнота обработки ошибочных данных; Наличие тестов для проверки допустимых значений входных данных; Наличие системы контроля полноты входных данных; Наличие средств контроля корректности входных данных; Наличие требований к программе по восстановлению процесса выполнения в случае сбоя ОС, внешних устройств, процессора; Наличие требований к программе по восстановлению результатов при отказах ОС, внешних устройств, процессора;

Наличие средств восстановления при сбоях оборудования; Наличие возможности повторного старта с точки прерывания Наличие обработки неопределенностей; Наличие централизованного управления процессами, конкурирующими из-за ресурсов; Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления;

*Оценочные элементы фактора «Сопровождаемость»*

Наличие комментариев в точках входа и выхода в программу; Осуществляется ли передача результатов работы модуля через вызывающий его модуль; Оценка программы по числу циклов;

Используется ли язык высокого уровня; Наличие проверки корректности передаваемых данных; Использовании при построении программ метода структурного программирования; Соблюдение принципа процесса разработки программы сверху вниз; Наличие ограничений на размеры модуля; Наличие модульной схемы программы.

*Оценочные элементы фактора «Корректность»*

Наличие всех необходимых документов для понимания и использования ПС; Наличие описание схемы иерархии модулей программы; Наличие описаний основных функций; Наличие описаний частных функций; Наличие описания данных; Наличие описания алгоритмов; Наличие описания интерфейсов между модулями; Наличие описания всех параметров; Наличие описание методов настройки системы; Наличие описание способов проверки работоспособности программы; Реанимация всех модулей системы; Реанимация всех основных функций; Реанимация всех алгоритмов; Наличие определений всех данных: переменные, индексы., массивы и пр. ; Наличие интерфейсов с пользователем; Отсутствие противоречий в выполнении основных функций; Отсутствие противоречий в выполнении частных функций; Отсутствие противоречий в выполнении алгоритмов; Правильность взаимосвязей; Правильность реализаций интерфейса с пользователем; Отсутствие противоречий в настройке системы;Комплектность документации соответствии со стандартами;

**5. На основании основных метрик описать функциональные особенности представленной в папке «экзамен» на рабочем столе программы под названием «5.exe». Определить основную цель приложения. Результат оформить в документе Word в виде таблицы.**

Метрики описания функциональных особенностей представленной программы могут включать:

1. Общее описание программы: Основной целью программы является...

2. Описание возможностей программы: Программа позволяет пользователю...

3. Описание основных функций программы: Программа включает следующие функции...

4. Описание алгоритмов и методов, используемых в программе: Для достижения своих функций, программа использует следующие алгоритмы и методы...

5. Описание интерфейса пользователя: Программа имеет удобный и интуитивно понятный интерфейс, позволяющий пользователю...

6. Описание входных и выходных данных: При запуске программы пользователь может предоставить следующие входные данные..., а программа возвращает следующие выходные данные...

7. Описание возможных ошибок и исключительных ситуаций: В случае возникновения ошибки, программа сообщает пользователю следующую информацию и предлагает возможные решения...

8. Описание требований к системе и зависимостей: Для работы программы необходима следующая системная конфигурация и зависимости...

9. Описание процесса установки и настройки программы: Чтобы установить программу, пользователь должен выполнить следующие шаги...

10. Описание поддержки и обновлений: Программа также предлагает следующую поддержку и возможности обновления...

*Оценочные элементы фактора «Надежность ПС»*

Наличие требований к программе по устойчивости функционирования при наличии ошибок во входных данных; Возможность обработки ошибочных ситуаций; Полнота обработки ошибочных данных; Наличие тестов для проверки допустимых значений входных данных; Наличие системы контроля полноты входных данных; Наличие средств контроля корректности входных данных; Наличие требований к программе по восстановлению процесса выполнения в случае сбоя ОС, внешних устройств, процессора; Наличие требований к программе по восстановлению результатов при отказах ОС, внешних устройств, процессора;

Наличие средств восстановления при сбоях оборудования; Наличие возможности повторного старта с точки прерывания Наличие обработки неопределенностей; Наличие централизованного управления процессами, конкурирующими из-за ресурсов; Наличие возможности автоматически обходить ошибочные ситуации в процессе вычисления;

*Оценочные элементы фактора «Сопровождаемость»*

Наличие комментариев в точках входа и выхода в программу; Осуществляется ли передача результатов работы модуля через вызывающий его модуль; Оценка программы по числу циклов;

Используется ли язык высокого уровня; Наличие проверки корректности передаваемых данных; Использовании при построении программ метода структурного программирования; Соблюдение принципа процесса разработки программы сверху вниз; Наличие ограничений на размеры модуля; Наличие модульной схемы программы.

*Оценочные элементы фактора «Корректность»*

Наличие всех необходимых документов для понимания и использования ПС; Наличие описание схемы иерархии модулей программы; Наличие описаний основных функций; Наличие описаний частных функций; Наличие описания данных; Наличие описания алгоритмов; Наличие описания интерфейсов между модулями; Наличие описания всех параметров; Наличие описание методов настройки системы; Наличие описание способов проверки работоспособности программы; Реанимация всех модулей системы; Реанимация всех основных функций; Реанимация всех алгоритмов; Наличие определений всех данных: переменные, индексы., массивы и пр. ; Наличие интерфейсов с пользователем; Отсутствие противоречий в выполнении основных функций; Отсутствие противоречий в выполнении частных функций; Отсутствие противоречий в выполнении алгоритмов; Правильность взаимосвязей; Правильность реализаций интерфейса с пользователем; Отсутствие противоречий в настройке системы;Комплектность документации соответствии со стандартами;

**6. Сравнить функциональные особенности представленных в папке «экзамен» на рабочем столе программ под названием «2.exe» и «5.exe». Определить основную цель приложений, описать отличительные черты каждого из них. Результат оформить в документе Word в виде таблицы.**

Сравнение функциональных особенностей программ является важным шагом при выборе подходящего программного обеспечения для конкретной задачи. Программы могут отличаться по своим функциональным возможностям, которые могут включать в себя:

6.1. Виды задач, которые программа может решать: одни программы могут быть специализированы на определенных задачах, например, фоторедактировании, аудиообработке или управлении проектами, в то время как другие программы могут предлагать широкий спектр функций для множества задач.

6.2. Интерфейс пользователя: разные программы могут иметь разные интерфейсы и способы взаимодействия с пользователем. Некоторые программы могут иметь интуитивно понятный интерфейс, легко настраиваемый под конкретные потребности пользователя, а другие могут предлагать более сложные и специализированные интерфейсы.

6.3. Функции и инструменты: программы могут предлагать различные функции и инструменты для работы с данными. Например, в фоторедакторе могут быть инструменты для обрезки, ретуширования и наложения фильтров на изображения, в то время как в программе управления проектами могут быть инструменты для планирования, распределения задач и мониторинга прогресса.

6.4. Интеграция с другими программами и системами: некоторые программы могут обеспечивать возможность интеграции с другими программами или системами, что позволяет обмениваться данными и использовать функционал других приложений. Например, программы управления задачами могут иметь возможность интеграции с электронной почтой или календарем.

**7. Сравнить функциональные особенности представленных в папке «экзамен» на рабочем столе программ под названием «4.exe» и «5.exe». Определить основную цель приложений, описать отличительные черты каждого из них. Результат оформить в документе Word в виде таблицы.**

Сравнение функциональных особенностей программ является важным шагом при выборе подходящего программного обеспечения для конкретной задачи. Программы могут отличаться по своим функциональным возможностям, которые могут включать в себя:

7.1. Виды задач, которые программа может решать: одни программы могут быть специализированы на определенных задачах, например, фоторедактировании, аудиообработке или управлении проектами, в то время как другие программы могут предлагать широкий спектр функций для множества задач.

7.2. Интерфейс пользователя: разные программы могут иметь разные интерфейсы и способы взаимодействия с пользователем. Некоторые программы могут иметь интуитивно понятный интерфейс, легко настраиваемый под конкретные потребности пользователя, а другие могут предлагать более сложные и специализированные интерфейсы.

7.3. Функции и инструменты: программы могут предлагать различные функции и инструменты для работы с данными. Например, в фоторедакторе могут быть инструменты для обрезки, ретуширования и наложения фильтров на изображения, в то время как в программе управления проектами могут быть инструменты для планирования, распределения задач и мониторинга прогресса.

7.4. Интеграция с другими программами и системами: некоторые программы могут обеспечивать возможность интеграции с другими программами или системами, что позволяет обмениваться данными и использовать функционал других приложений. Например, программы управления задачами могут иметь возможность интеграции с электронной почтой или календарем.

**8. Сравнить функциональные особенности представленных в папке «экзамен» на рабочем столе программ под названием «2.exe» и «3.exe». Определить основную цель приложений, описать отличительные черты каждого из них. Результат оформить в документе Word в виде таблицы.**

Сравнение функциональных особенностей программ является важным шагом при выборе подходящего программного обеспечения для конкретной задачи. Программы могут отличаться по своим функциональным возможностям, которые могут включать в себя:

8.1. Виды задач, которые программа может решать: одни программы могут быть специализированы на определенных задачах, например, фоторедактировании, аудиообработке или управлении проектами, в то время как другие программы могут предлагать широкий спектр функций для множества задач.

8.2. Интерфейс пользователя: разные программы могут иметь разные интерфейсы и способы взаимодействия с пользователем. Некоторые программы могут иметь интуитивно понятный интерфейс, легко настраиваемый под конкретные потребности пользователя, а другие могут предлагать более сложные и специализированные интерфейсы.

8.3. Функции и инструменты: программы могут предлагать различные функции и инструменты для работы с данными. Например, в фоторедакторе могут быть инструменты для обрезки, ретуширования и наложения фильтров на изображения, в то время как в программе управления проектами могут быть инструменты для планирования, распределения задач и мониторинга прогресса.

8.4. Интеграция с другими программами и системами: некоторые программы могут обеспечивать возможность интеграции с другими программами или системами, что позволяет обмениваться данными и использовать функционал других приложений. Например, программы управления задачами могут иметь возможность интеграции с электронной почтой или календарем.

**9. Сравнить функциональные особенности представленных в папке «экзамен» на рабочем столе программ под названием «4.exe» и «3.exe». Определить основную цель приложений, описать отличительные черты каждого из них. Результат оформить в документе Word в виде таблицы.**

Сравнение функциональных особенностей программ является важным шагом при выборе подходящего программного обеспечения для конкретной задачи. Программы могут отличаться по своим функциональным возможностям, которые могут включать в себя:

9.1. Виды задач, которые программа может решать: одни программы могут быть специализированы на определенных задачах, например, фоторедактировании, аудиообработке или управлении проектами, в то время как другие программы могут предлагать широкий спектр функций для множества задач.

9.2. Интерфейс пользователя: разные программы могут иметь разные интерфейсы и способы взаимодействия с пользователем. Некоторые программы могут иметь интуитивно понятный интерфейс, легко настраиваемый под конкретные потребности пользователя, а другие могут предлагать более сложные и специализированные интерфейсы.

9.3. Функции и инструменты: программы могут предлагать различные функции и инструменты для работы с данными. Например, в фоторедакторе могут быть инструменты для обрезки, ретуширования и наложения фильтров на изображения, в то время как в программе управления проектами могут быть инструменты для планирования, распределения задач и мониторинга прогресса.

9.4. Интеграция с другими программами и системами: некоторые программы могут обеспечивать возможность интеграции с другими программами или системами, что позволяет обмениваться данными и использовать функционал других приложений. Например, программы управления задачами могут иметь возможность интеграции с электронной почтой или календарем.

**10. Ввести в действие пять ветвей разработок в двух репозиториях, переместить в подкаталог.**

Чтобы в GitHub создать и перенести пять веток разработок в двух репозиториях и переместить их в подкаталог, вам потребуется выполнить следующие шаги:

10.1. Создание репозитория:

- Зайдите на страницу GitHub и авторизуйтесь в своем аккаунте.

- Нажмите кнопку "New repository" (Создать репозиторий) в правом верхнем углу.

- Задайте имя репозитория и выберите другие настройки по вашему желанию.

- Нажмите кнопку "Create repository" (Создать репозиторий) для создания нового репозитория.

- Повторите эти шаги, чтобы создать второй репозиторий.

10.2. Клонирование репозиториев:

- Откройте Git Bash или любой другой терминал, где вы можете выполнять Git-команды.

- Перейдите в директорию, в которую вы хотите клонировать репозиторий. Например, используйте команду "cd /путь/к/директории".

- Склонируйте первый репозиторий при помощи команды "git clone <URL первого репозитория>".

- Повторите этот шаг для второго репозитория.

10.3. Создание и переключение на новые ветки:

- Перейдите в директорию первого клонированного репозитория с помощью команды "cd <имя директории>".

- Создайте пять новых веток при помощи команды "git branch <имя ветки>" для каждой ветки. Например, "git branch branch1".

- Переключитесь на каждую созданную ветку при помощи команды "git checkout <имя ветки>". Например, "git checkout branch1".

- Повторите эти шаги для второго клонированного репозитория.

10.4. Перемещение веток в подкаталог:

- В директории первого клонированного репозитория выполните команду "mkdir <имя подкаталога>" для создания подкаталога. Например, "mkdir subdirectory".

- Переместите каждую созданную ветку в подкаталог с помощью команды "git branch -m <старое имя ветки> <имя подкаталога>/<новое имя ветки>". Например, "git branch -m branch1 subdirectory/branch1".

- Повторите эти шаги для второго клонированного репозитория.

10.5. Загрузка изменений на GitHub:

- Выполните команду "git push --all" для каждого клонированного репозитория, чтобы загрузить новые ветки и изменения на GitHub.

Теперь у вас должны быть созданы и перемещены пять веток разработки в подкаталог в каждом из двух репозиториев на GitHub.

**11. Переместить Branch1 из исходного репозитория в соседний, добавить изменения и применить команду «Push», и просмотреть по системе контроля версий, дату обновления файла Branch1.**

Для перемещения Branch1 из исходного репозитория в соседний можно использовать следующие шаги:

1. Сначала склонируйте исходный репозиторий в локальную директорию на вашем компьютере:

```

git clone <URL исходного репозитория>

```

2. Перейдите в склонированный репозиторий:

```

cd <директория репозитория>

```

3. Создайте новый репозиторий в соседней директории или используйте уже существующий пустой репозиторий. Если нужно создать новый репозиторий, выполните команду:

```

git init --bare <путь к новому репозиторию>

```

4. Перейдите в исходный репозиторий с помощью команды "cd":

```

cd ..

cd <директория исходного репозитория>

```

5. Переместите Branch1 в новый репозиторий с помощью команды "git push":

```

git push --mirror <путь к новому репозиторию>

```

6. Перейдите в директорию нового репозитория:

```

cd <директория нового репозитория>

```

7. Внесите изменения в Branch1, как вам нужно.

8. Затем примените команду "git push" для отправки изменений в удаленный репозиторий:

```

git push origin Branch1

```

Посмотреть дату обновления файла Branch1 можно с помощью системы контроля версий, например, командой "git log":

```

git log --pretty=format:"%ad" --branches=<путь к файлу Branch1>

```

Это позволит вам увидеть даты всех коммитов, в том числе и коммитов изменений файла Branch1.

**12. Извлечь изменения из удалённого репозитория и применить их к текущей ветви с помощью команды «Git Pull».**

Для извлечения изменений из удалённого репозитория и их применения к текущей ветви с помощью команды «Git Pull» выполните следующие шаги:

1. Откройте командную строку или терминал в директории вашего локального репозитория.

2. Убедитесь, что вы находитесь на нужной ветви, на которую хотите применить изменения. Для этого вы можете использовать команду `git branch`, чтобы посмотреть список доступных веток, и `git checkout <branch\_name>`, чтобы переключиться на нужную ветвь.

3. Затем используйте команду `git pull <remote\_name> <branch\_name>`, где `<remote\_name>` - название удалённого репозитория, а `<branch\_name>` - название ветви, из которой вы хотите извлечь изменения.

Например, если удалённый репозиторий называется «origin» и вы хотите извлечь изменения из ветви «master», команда будет выглядеть так: `git pull origin master`.

4. Git выполнит операцию слияния (merge) и применит изменения из удалённого репозитория к вашей текущей ветви.

Если появятся конфликты слияния, Git сообщит вам об этом и остановится, чтобы вы могли разрешить конфликты вручную.

5. После успешного слияния изменений вы можете продолжить работу с вашей ветвью и выполнить команду `git push`, чтобы отправить изменения в удалённый репозиторий.

**13. Синхронизировать финальную версию предлагаемого проекта с репозиторием на GitHub.**

Для синхронизации финальной версии предлагаемого проекта с репозиторием на GitHub, следуйте этим шагам:

1. Создайте новый репозиторий на GitHub, если его еще нет.

2. Откройте терминал (или командную строку) на вашем компьютере и перейдите в папку проекта с помощью команды `cd путь\_к\_папке\_проекта`.

3. Инициализируйте репозиторий Git в папке проекта с помощью команды `git init`. Это создаст пустой репозиторий Git в вашей локальной папке проекта.

4. Добавьте все файлы проекта в индекс с помощью команды `git add .`. Это добавит все файлы в вашем проекте в индекс Git для отслеживания изменений.

5. Создайте коммит, используя команду `git commit -m "Ваше\_сообщение\_коммита"`. Здесь вы можете ввести сообщение коммита, описывающее изменения, которые вы внесли.

6. На GitHub, скопируйте URL-адрес вашего репозитория (вида `https://github.com/ваш\_пользователь/ваш\_репозиторий.git`).

7. Добавьте удаленный репозиторий к вашему локальному репозиторию с помощью команды `git remote add origin URL-адрес\_удаленного\_репозитория`. Здесь URL-адрес\_удаленного\_репозитория - это скопированный URL-адрес в шаге 6.

8. Загрузите ваши коммиты (вместе с финальной версией проекта) на GitHub с помощью команды `git push -u origin master`. Это отправит все ваши коммиты на удаленный репозиторий на GitHub.

Теперь ваша финальная версия проекта должна быть синхронизирована с репозиторием на GitHub. Вы сможете увидеть все изменения и коммиты на странице репозитория на GitHub.

**14. Объединить ветви «final» и «master» из двух разных репозиториев, представленных удалённо.**

ля объединения ветвей "final" и "master" из двух разных репозиториев, представленных удаленно, можно использовать команду git pull с опцией --allow-unrelated-histories.

1. Подключитесь к одному из репозиториев, например, к репозиторию с ветвью "master".

```

git remote add repo1 <URL\_репозитория\_1>

git fetch repo1

```

2. Переключитесь на ветвь "master".

```

git checkout master

```

3. Объедините изменения из ветви "final" в ветвь "master".

```

git pull repo1 final --allow-unrelated-histories

```

4. Повторите шаги 1-3 для второго репозитория, используя другое имя для подключения удаленного репозитория (например, repo2).

5. Если требуется, решите возможные конфликты, которые могут возникнуть при объединении ветвей.

6. Закрепите изменения на удаленном репозитории.

```

git push origin master

```

Теперь ветви "final" и "master" из разных репозиториев объединены в ветви "master" удаленного репозитория.

**15. Разработать в репозитории с помощью функции «menu» две ветви проекта.**

Чтобы создать ветви проекта в репозитории с помощью функции "menu", нужно выполнить следующие шаги:

1. Инициализировать репозиторий: В командной строке или в среде разработки выполните команду `git init` в папке проекта. Это создаст новый репозиторий Git в этой папке.

2. Добавить файлы в репозиторий: Поместите все файлы вашего проекта в папку репозитория. Вы можете использовать команду `git add .` для добавления всех файлов в репозиторий или `git add <имя\_файла>` для добавления отдельных файлов.

3. Создать основную ветвь: Выполните команду `git branch main` для создания основной ветви проекта. Она будет содержать стабильную версию вашего проекта.

4. Создать вторую ветвь: Выполните команду `git branch feature` для создания второй ветви проекта. Эта ветвь будет использоваться для разработки новых функций или исправлений ошибок.

5. Переключиться на вторую ветвь: Выполните команду `git checkout feature` для переключения на вторую ветвь проекта.

6. Внести изменения в проект: Внесите необходимые изменения в своем проекте. Добавьте новые функции или исправьте ошибки.

7. Закоммить изменения: Когда вы закончите выполнение изменений во второй ветви, выполните команду `git commit -m "Описание ваших изменений"`, чтобы закоммитить изменения в ваш репозиторий.

8. Переключиться на основную ветвь: Выполните команду `git checkout main` для переключения обратно на основную ветвь проекта.

9. Объединить изменения: Выполните команду `git merge feature` для объединения ваших изменений из второй ветви в основную ветвь.

Теперь в вашем репозитории есть две ветви проекта - основная и вторая ветвь, которая содержит новые функции или исправления ошибок. Вы можете продолжить работать в одной из них или создать дополнительные ветви, если это необходимо.

**16. В программе IDA pro 8.0 дизассемблировать файл «6.exe», находящийся в папке «экзамен» на рабочем столе. Отразить в виде таблицы в MS Word библиотеки, к которым он обращается, и указать, где начинается работа программы.**

Для этого выполните следующие шаги:

1. Откройте файл для дизассемблирования в IDA Pro.

2. В меню IDA Pro выберите "Загрузить файл" (File -> Load File) и выберите соответствующий файл.

3. После загрузки файла нажмите "Анализ" (Analysis) -> "Анализировать выбранную область" (Analyze Selection) или использовать горячую клавишу "A".

4. После завершения анализа перейдите в окно "Кросс-эффекты" (Cross-References) путем выбора "Окно" (View) -> "Окна выбранного типа" (Open Subviews) -> "Кросс-эффекты" (Cross-References).

5. В окне "Кросс-эффекты" найдите раздел "Библиотеки" (Libraries) или "Динамические функции" (Dynamic Functions). В этом разделе вы увидите вызываемые функции или библиотеки, на которые ссылается программа.

6. Чтобы исследовать более подробную информацию о вызываемой функции или библиотеке, щелкните правой кнопкой мыши на соответствующей записи и выберите "Перейти к определению" (Go to Definition) или "Разобрать" (Disassemble).

Теперь вы можете исследовать вызываемые библиотеки и функции в программе, используя IDA Pro.

**17. Создать хранилища (репозитория) SVN и разместить в нем каталог. Выполнить импорт данных (Tortoise SVN).**

1. Выберите "File" (Файл) в качестве типа хранилища и введите имя для хранилища. Например, "my\_repository" (мое\_хранилище). Нажмите "OK" (ОК), чтобы завершить создание репозитория SVN.

2. Теперь в созданной папке хранилища "my\_repository" создайте каталог, который вы хотите импортировать в SVN.

3. Щелкните правой кнопкой мыши на созданном каталоге и выберите "TortoiseSVN" и затем "Import" (Импорт) в контекстном меню.

4. В диалоговом окне "URL of repository" (URL хранилища) введите путь к вашему хранилищу SVN. Например, "file:///C:/path/to/my\_repository" (файл:///C:/путь/к/моему\_хранилищу). Нажмите "OK" (ОК), чтобы перейти к следующему шагу.

5. В появившемся диалоговом окне "Import" (Импорт) укажите "URL" (URL), оставьте "Empty log message" (Пустое сообщение в журнале) и убедитесь, что "HEAD revision" (HEAD ревизия) выбрана. Нажмите "OK" (ОК), чтобы начать импорт данных.

6. Tortoise SVN выполнит импорт выбранного каталога в ваше хранилище SVN.

Теперь ваш каталог успешно импортирован в хранилище SVN. Вы можете использовать Tortoise SVN для управления версиями, коммита, обновления и других операций над вашим проектом.

**18. Установить параметры работы среды Visual Studio, соответствующие Web-разработке.**

1. Откройте Visual Studio.

2. Нажмите на "Инструменты" в верхнем меню, затем выберите "Параметры".

3. В появившемся окне выберите "Среда разработки" в левой панели.

4. Настройте следующие параметры:

- Общие:

- "Писать в файл журнала" - установите галочку, чтобы сохранять журнал событий.

- "Вывод журнала" - выберите нужный формат вывода для журнала событий.

- "Данные профайлера" - укажите путь для сохранения данных профайлера.

- "Уведомления о доступности обновлений" - выберите соответствующую опцию уведомлений о доступности обновлений Visual Studio.

- Конфигурации редактора:

- "Отчет Unicode включить" - установите галочку, чтобы использовать Unicode в проектах.

- "Расположение отсчета числа строк" - выберите соответствующую опцию для отображения номеров строк.

- "Табуляция" - выберите нужное количество пробелов для замены табуляции.

- "Отображение пробельных символов" - установите галочку, чтобы отображать пробельные символы в редакторе.

- Веб-разработка:

- "Использовать браузер" - выберите предпочитаемый браузер для просмотра веб-приложений.

- "Версия .NET Core" - выберите версию .NET Core, которую хотите использовать.

- "Использовать встроенный IIS Express" - установите галочку, чтобы использовать встроенный IIS Express для отладки веб-приложений.

5. Нажмите "OK", чтобы сохранить изменения.

Теперь ваша среда Visual Studio настроена для работы с веб-разработкой.

**19. В Visual Studio импортировать настройки из файла в папке «Экзамен» на рабочем столе, в который выгружены настройки.**

Для импорта настроек из файла в Visual Studio можно воспользоваться следующими шагами:

1. Откройте Visual Studio и перейдите в раздел "Средства" (Tools) в главном меню.

2. Выберите "Настройки" (Options) в выпадающем меню "Средства" (Tools).

3. В окне "Настройки" (Options) выберите "Настройки среды" (Environment) в левой панели.

4. В правой панели найдите раздел "Импорт и экспорт" (Import and Export).

5. Нажмите на кнопку "Импортировать/Экспортировать настройки" (Import and Export Settings).

6. В появившемся окне выберите "Импортировать настройки среды" (Import selected environment settings) и нажмите на кнопку "Далее" (Next).

7. В следующем окне выберите "Браузер" (Browse).

8. В открывшемся окне навигации найдите папку "Экзамен" на рабочем столе (или введите путь к ней вручную) и выберите файл настроек для импорта.

9. Нажмите на кнопку "Открыть" (Open).

10. После выбора файла настроек нажмите на кнопку "Закрыть" (Finish).

Теперь выбранные настройки должны быть импортированы в Visual Studio.

Обратите внимание, что эти инструкции основаны на Visual Studio 2019. Пути и названия элементов интерфейса могут отличаться для других версий Visual Studio.

**20. В программе IDA pro 8.0 дизассемблировать файл \*\*\*.exe. Отобразить главную функцию работы программы. Для функции Start выделить функции sub,call и push.**

Чтобы дизассемблировать файл .exe с помощью программы IDA Pro 8.0 и отобразить главную функцию, вам нужно выполнить следующие шаги:

1. Откройте программу IDA Pro 8.0.

2. В главном меню выберите "File" (Файл) -> "Open" (Открыть) или используйте горячие клавиши Ctrl+O.

3. В диалоговом окне выберите ваш файл .exe и нажмите "Open" (Открыть).

4. IDA Pro начнет анализировать файл. Это может занять некоторое время в зависимости от размера и сложности файла.

5. По завершении анализа IDA Pro отобразит дизассемблированный код программы.

6. Чтобы найти главную функцию, воспользуйтесь окном "Functions" (Функции), которое обычно находится слева.

7. В окне "Functions" найдите функцию, которая содержит основной код программы, например, функцию с именем "main" или "WinMain" (для Windows-приложений). Это может быть первая функция, вызванная при запуске программы.

8. Щелкните правой кнопкой мыши на этой функции и выберите "Jump to xref(s)" (Перейти к ссылкам) или используйте горячие клавиши Ctrl+X, чтобы перейти к этой функции.

Чтобы выделить внутри функции sub, call и push, вам нужно выполнить следующие действия:

1. Выделите несколько инструкций, которые вы считаете соответствующими функции sub. Обычно это инструкции с операцией вычитания (sub).

2. Выделите несколько инструкций, которые вы считаете соответствующими функции call. Обычно это инструкции с операцией вызова функции (call).

3. Выделите несколько инструкций, которые вы считаете соответствующими функции push. Обычно это инструкции с операцией помещения значения в стек (push).

Обратите внимание, что IDA Pro может идентифицировать основные блоки кода автоматически, и вы можете использовать это для определения границ функций. Если IDA Pro неправильно определил границы функций или вы хотите вручную изменить границы функции, вы можете щелкнуть правой кнопкой мыши в режиме графа и выбрать "Edit Function" (Редактировать функцию), чтобы вручную нарисовать границы функции.

Надеюсь, эта информация поможет вам дизассемблировать файл .exe в программе IDA Pro 8.0 и отобразить главную функцию работы программы, а также выделить функции sub, call и push внутри функции Start.

**21. Создать более двух ветвей в локальном репозитории, и соединить с глобальным репозиторием GitHub и настройкой доступа к каждой ветке.**

Для создания более двух ветвей в локальном репозитории и их соединения с глобальным репозиторием на GitHub, а также настройки доступа к каждой ветке, следуйте этим шагам:

1. Создайте локальный репозиторий:

- Откройте командную строку или терминал на вашем компьютере.

- Создайте новую папку для вашего репозитория с помощью команды `mkdir <название папки>`.

- Перейдите в эту новую папку с помощью команды `cd <название папки>`.

- Инициализируйте новый репозиторий с помощью команды `git init`.

2. Создайте первую ветвь:

- Создайте новую ветвь с помощью команды `git branch <название ветви>`. Например, `git branch branch1`.

- Переключитесь на эту новую ветвь с помощью команды `git checkout <название ветви>`. Например, `git checkout branch1`.

- Добавьте, измените и зафиксируйте некоторые файлы в вашем локальном репозитории.

3. Создайте вторую ветвь:

- Вернитесь в основную ветвь с помощью команды `git checkout master`.

- Создайте еще одну новую ветвь с помощью команды `git branch <название ветви>`. Например, `git branch branch2`.

- Переключитесь на эту новую ветвь с помощью команды `git checkout <название ветви>`. Например, `git checkout branch2`.

- Добавьте, измените и зафиксируйте некоторые файлы в вашем локальном репозитории.

4. Создайте удаленный репозиторий на GitHub:

- Откройте GitHub веб-сайт и войдите в свою учетную запись.

- Нажмите на зеленую кнопку "New" вверху страницы, чтобы создать новый репозиторий.

- Введите имя репозитория и выберите опции для его настройки, такие как описание, .gitignore файлы и лицензия.

- Нажмите на зеленую кнопку "Create repository", чтобы создать удаленный репозиторий.

5. Свяжите локальный репозиторий с удаленным репозиторием на GitHub:

- Скопируйте URL удаленного репозитория на GitHub.

- В локальном репозитории, в командной строке или терминале, выполните команду `git remote add origin <URL удаленного репозитория>`. Например, `git remote add origin https://github.com/your-username/your-repository.git`.

- Отправьте все ветки и коммиты из локального репозитория в удаленный репозиторий на GitHub, выполнив команду `git push -u origin --all`.

6. Настройте доступ к каждой ветке:

- На веб-сайте GitHub откройте свой удаленный репозиторий.

- Нажмите на вкладку "Settings" вверху страницы, затем выберите вкладку "Branches".

- Для каждой ветви выберите "Protect" и настройте правила доступа в соответствии с вашими предпочтениями. Например, вы можете выбрать, чтобы только вы и коллеги имели доступ к некоторым ветвям, или предоставить доступ для некоторых ветвей всем участникам вашего репозитория.

Теперь у вас есть локальный репозиторий с несколькими ветвями, который связан с глобальным репозиторием на GitHub, а каждая ветвь имеет настроенные правила доступа.

**22. Создать новую ветку СКВ с помощью команды Git Branch, перейти в репозиторий и создать новый текстовый файл.**

Для создания новой ветки в Git используется команда `git branch`. Предположим, что репозиторий уже существует и мы находимся в его корневой директории. Давайте создадим новую ветку, назовем ее "feature/new-branch", и перейдем в нее:

1. Откройте терминал или командную строку.

2. Перейдите в директорию репозитория с помощью команды `cd /path/to/repository`.

3. Выполните команду `git branch feature/new-branch` для создания новой ветки.

Примечание: Если вы хотите создать ветку отличную от текущей ветки, выполните команду `git branch feature/new-branch origin/other-branch`, где `other-branch` - имя существующей ветки.

4. Выполните команду `git checkout feature/new-branch` для перехода в новую ветку.

Теперь, когда мы находимся в новой ветке, мы можем создать новый текстовый файл:

1. Выполните команду `touch new-file.txt`, чтобы создать новый пустой текстовый файл.

2. Редактируйте файл по своему усмотрению, например, с помощью команды `nano new-file.txt`.

3. Сохраните изменения и закройте редактор.

Теперь у вас есть новая ветка в вашем репозитории и новый текстовый файл в этой ветке.

**23. Создать локальный репозиторий, связанный с заданным глобальным репозиторием. Настроить параметры доступа к локальному репозиторию GitHub.**

заданным глобальным репозиторием на GitHub, выполните следующие шаги:

1. Установите Git на свой компьютер, если он еще не установлен. Вы можете найти инструкции по установке Git на официальном сайте Git (https://git-scm.com/downloads).

2. Откройте командную строку или терминал на вашем компьютере.

3. Перейдите в папку, где вы хотите создать локальный репозиторий.

4. Склонируйте глобальный репозиторий с GitHub с помощью команды `git clone <URL репозитория>`, где `<URL репозитория>` - ссылка на глобальный репозиторий. Например:

```

git clone https://github.com/username/repository.git

```

Примечание: Замените "username" и "repository" на соответствующие значения своего глобального репозитория на GitHub.

5. Перейдите в склонированную папку репозитория с помощью команды `cd repository`, где "repository" - имя папки, созданной после клонирования репозитория.

6. Настройте параметры доступа к локальному репозиторию GitHub, чтобы вы могли работать с ним. Чтобы это сделать, выполните команды:

```

git config --global user.name "Ваше имя"

git config --global user.email "Ваш email"

```

Убедитесь, что вы указали свои реальное имя и email.

Теперь у вас есть локальный репозиторий, связанный с заданным глобальным репозиторием на GitHub, и вы можете начать работу с ним.

**24. Выполнить загрузку изменений выполненных в локальном репозитории в глобальный репозитарий GitHub.**

Для выполнения загрузки изменений из локального репозитория в глобальный репозиторий GitHub следуйте этим шагам:

1. Во-первых, убедитесь, что у вас уже есть глобальный репозиторий на GitHub, и вы имеете доступ к нему.

2. Откройте командную строку или терминал на вашем компьютере.

3. Перейдите в локальную директорию вашего репозитория с помощью команды "cd".

4. Если еще не настроены глобальные настройки пользователя для Git, выполните следующие команды, указав свое имя пользователя и адрес электронной почты:

```

git config --global user.name "Ваше имя пользователя"

git config --global user.email "ваш@адрес.электронной.почты"

```

5. Далее, свяжите ваш локальный репозиторий с глобальным репозиторием на GitHub с помощью команды:

```

git remote add origin [URL-адрес вашего глобального репозитория на GitHub]

```

6. Выполните команду "git remote -v", чтобы убедиться, что связь между локальным и глобальным репозиториями настроена правильно.

7. Сделайте коммит локальных изменений для подготовки их к загрузке с помощью команды:

```

git commit -m "Описание ваших изменений"

```

8. Выполните команду "git push origin [название вашей ветки]", чтобы загрузить ваши изменения из локального репозитория в глобальный репозиторий на GitHub.

9. Введите свои учетные данные для входа в GitHub, если вам запросят это сделать.

10. После успешной загрузки изменений вы увидите их в глобальном репозитории на GitHub.

Это все! Теперь вы загрузили изменения из локального репозитория в глобальный репозиторий на GitHub.

**25. Создать репозиторий на сервисе GitHub. Создать несколько типов организаций с различными параметрами доступа.**

1. Организация с открытым доступом:

- Название: OpenOrg

- Публичный доступ для всех пользователей GitHub

- Все пользователи могут видеть и клонировать репозиторий

2. Организация с ограниченным доступом:

- Название: RestrictedOrg

- Частный доступ

- Только участники организации могут видеть и клонировать репозиторий

3. Организация с ограниченным доступом и разрешением на изменение:

- Название: EditOrg

- Частный доступ

- Только участники организации могут видеть и клонировать репозиторий

- Только участники организации с правами на изменения могут модифицировать репозиторий

Для создания репозитория на GitHub, выполните следующие шаги:

1. Авторизуйтесь на GitHub и перейдите на страницу вашего профиля.

2. В правом верхнем углу страницы нажмите кнопку "New".

3. Заполните поле "Repository name" указав желаемое название репозитория.

4. Выберите опцию "Public" для открытого доступа или "Private" для ограниченного доступа.

5. Если создается организация, отметьте флажок "Initialize this repository with a README" (при необходимости).

6. Нажмите кнопку "Create repository".

После создания репозитория, вы можете пригласить других пользователей в организацию и настроить их уровень доступа к репозиторию (читатель, разработчик, администратор).

**26. Создать несколько учетных записей для разработчиков. Задать параметры, необходимые для отправки и слияния копии репозитория.**

Для создания учетных записей разработчиков, необходимо выполнить следующие шаги:

1. Открыть центр администрирования репозитория (например, GitHub или Bitbucket).

2. Нажмите на "Создать новую учетную запись".

3. Заполните необходимые данные для каждой учетной записи, такие как:

- Имя пользователя: выберите уникальное имя для каждого разработчика.

- Пароль: задайте пароль для каждой учетной записи.

- Адрес электронной почты: укажите адрес электронной почты каждого разработчика.

- Роль: выберите роль "Разработчик" или "Коммитер" для каждого пользователя. Разработчики имеют права на отправку изменений, а коммитеры также имеют права на слияние изменений.

4. Нажмите на кнопку "Создать учетную запись" или аналогичную кнопку для завершения создания учетной записи.

Чтобы задать параметры, необходимые для отправки и слияния копии репозитория, выполните следующие действия:

1. Откройте репозиторий, в котором будут настроены параметры.

2. Перейдите в раздел "Настройки репозитория" или аналогичный раздел.

3. Найдите параметры "Доступ" или "Управление доступом".

4. Добавьте учетные записи разработчиков, которым требуется разрешение на отправку изменений.

5. Для каждой учетной записи разработчика установите разрешение "Запись" или аналогичное разрешение, чтобы дать разработчикам возможность отправлять изменения в репозиторий.

6. Если вы хотите, чтобы определенные разработчики имели права на слияние изменений, в разрешениях учетной записи коммитера установите "Слияние" или аналогичное разрешение.

После задания параметров разработчики смогут отправить изменения в репозиторий и, если им разрешено, выполнить слияние изменений.

**27. Создать приложение MVC в среде Visual Studio, обеспечивающее ввод данных о некотором объекте.**

Вот пример создания приложения MVC в среде Visual Studio для ввода данных о некотором объекте (например, о студенте).

Шаги:

1. Создайте новый проект в Visual Studio с использованием шаблона ASP.NET MVC приложения.

2. В Solution Explorer щелкните правой кнопкой мыши на проекте (не на папке) и выберите "Add" -> "Controller".

3. В появившемся окне выберите шаблон "Empty MVC Controller" и назовите его, например, "StudentController".

4. Внутри контроллера добавьте следующий метод:

```csharp

public ActionResult Create()

{

return View();

}

```

5. Создайте представление для этого метода. Щелкните правой кнопкой мыши на папке "Views" -> "Student" и выберите "Add" -> "View".

6. В появившемся окне выберите параметр "Create" и нажмите "Add". Это создаст представление с именем "Create.cshtml".

7. Отредактируйте представление "Create.cshtml" следующим образом:

```html

@model StudentModel // здесь StudentModel - это модель данных вашего объекта

@{

ViewBag.Title = "Create";

}

<h2>Create Student</h2>

@using (Html.BeginForm())

{

<div>

@Html.LabelFor(model => model.Name)

@Html.TextBoxFor(model => model.Name)

@Html.ValidationMessageFor(model => model.Name)

</div>

<div>

@Html.LabelFor(model => model.Age)

@Html.TextBoxFor(model => model.Age)

@Html.ValidationMessageFor(model => model.Age)

</div>

<input type="submit" value="Save" />

}

```

8. Создайте модель для вашего объекта. Щелкните правой кнопкой мыши на папке "Models" и выберите "Add" -> "Class".

9. Назовите модель, например, "StudentModel", и определите в ней свойства объекта (например, имя и возраст):

```csharp

public class StudentModel

{

public string Name { get; set; }

public int Age { get; set; }

}

```

10. В контроллере "StudentController" добавьте POST-метод для сохранения данных:

```csharp

[HttpPost]

public ActionResult Create(StudentModel student)

{

// Ваш код для сохранения данных о студенте

return RedirectToAction("Index", "Home");

}

```

11. Запустите приложение и перейдите по адресу "/Student/Create".

**28. Проверьте правильность ввода данных с помощью отрицательного и положительного теста на стороне клиента.**

Пример проверки ввода данных на стороне клиента с использованием отрицательного и положительного теста:

```csharp

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main(string[] args)

{

// Отрицательный тест - проверяем наличие цифр

string input1 = "abc123";

bool isInput1Valid = ValidateInput(input1);

Console.WriteLine("Отрицательный тест: " + isInput1Valid);

// Положительный тест - только буквы

string input2 = "abc";

bool isInput2Valid = ValidateInput(input2);

Console.WriteLine("Положительный тест: " + isInput2Valid);

Console.ReadLine();

}

static bool ValidateInput(string input)

{

// Регулярное выражение для проверки наличия только букв

string pattern = "^[a-zA-Z]+$";

// Проверка с использованием регулярного выражения

bool isInputValid = Regex.IsMatch(input, pattern);

return isInputValid;

}

}

```

В этом примере вводимые данные проверяются на наличие только букв в положительном тесте и проверяются наличие цифр в отрицательном тесте. Валидацию данных осуществляет метод `ValidateInput`, который использует регулярное выражение для проверки.

Возможные результаты выполнения:

```

Отрицательный тест: False

Положительный тест: True

```

В данном случае, отрицательный тест не прошел проверку, так как ввод содержит цифры. Положительный тест прошел проверку, так как ввод состоит только из букв.

**29. Создать сервер (TCP/IP) в среде Visual Studio на языке C#.**

Для создания сервера TCP/IP в среде Visual Studio на языке C# необходимо использовать классы из пространства имён System.Net.Sockets.

Приведу пример кода, который создаст TCP/IP сервер, принимающий подключения от клиентов и отправляющий им эхо-сообщения:

```csharp

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class TCPServer

{

static void Main()

{

// Создаем объект TcpListener и задаем адрес и порт для прослушивания

IPAddress ipAddress = IPAddress.Parse("127.0.0.1");

int port = 12345;

TcpListener server = new TcpListener(ipAddress, port);

try

{

// Запускаем сервер

server.Start();

Console.WriteLine("Сервер запущен. Ожидание подключений...");

// Принимаем подключения в бесконечном цикле

while (true)

{

// Принимаем подключение от клиента

TcpClient client = server.AcceptTcpClient();

Console.WriteLine("Подключен клиент: {0}", client.Client.RemoteEndPoint);

// Получаем поток сетевых данных для чтения/записи

NetworkStream stream = client.GetStream();

// Создаем буфер для чтения данных

byte[] buffer = new byte[client.ReceiveBufferSize];

// Читаем данные от клиента в бесконечном цикле

while (true)

{

// Читаем данные из потока

int bytesRead = stream.Read(buffer, 0, buffer.Length);

// Преобразуем байты в строку

string dataReceived = Encoding.UTF8.GetString(buffer, 0, bytesRead);

Console.WriteLine("Получено сообщение от клиента: {0}", dataReceived);

// Преобразуем строку в байты

byte[] dataToSend = Encoding.UTF8.GetBytes(dataReceived);

// Отправляем эхо-сообщение клиенту

stream.Write(dataToSend, 0, dataToSend.Length);

}

}

}

catch (Exception ex)

{

Console.WriteLine("Ошибка: " + ex.Message);

}

finally

{

// Останавливаем сервер

server.Stop();

}

}

}

```

Не забудьте изменить IP-адрес (в данном случае указано "127.0.0.1") и порт (в данном случае указано "12345") под свои нужды.

Этот код создаст TCP/IP сервер, который будет ожидать подключений от клиентов и отправлять им эхо-сообщения, которые будут совпадать с отправленными данными от клиента.

**30. Реализовать на сервере проверку данных (TCP/IP) в среде Visual Studio на языке C#.**

Для реализации проверки данных на сервере в среде Visual Studio на языке C# можно использовать класс TcpListener из пространства имен System.Net.Sockets.

Ниже приведен пример кода, демонстрирующий базовую реализацию проверки данных на сервере:

```csharp

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

class Program

{

static void Main(string[] args)

{

// Устанавливаем IP-адрес и порт для прослушивания

IPAddress ipAddress = IPAddress.Parse("127.0.0.1");

int port = 1234;

// Создаем объект TcpListener и передаем ему IP-адрес и порт

TcpListener server = new TcpListener(ipAddress, port);

// Начинаем прослушивание

server.Start();

Console.WriteLine("Сервер запущен...");

while (true)

{

// Принимаем входящее подключение от клиента

TcpClient client = server.AcceptTcpClient();

Console.WriteLine("Подключение принято.");

// Получаем сетевой поток для чтения и записи

NetworkStream stream = client.GetStream();

byte[] data = new byte[client.ReceiveBufferSize];

int bytesRead = stream.Read(data, 0, data.Length);

// Преобразуем полученные данные в строку

string receivedMessage = Encoding.ASCII.GetString(data, 0, bytesRead);

Console.WriteLine("Получены данные: " + receivedMessage);

// Проверяем данные на сервере

bool isValid = CheckData(receivedMessage);

// Формируем ответное сообщение

string responseMessage = isValid ? "Данные прошли проверку." : "Данные не прошли проверку.";

// Преобразуем строку ответа в массив байтов

byte[] response = Encoding.ASCII.GetBytes(responseMessage);

// Отправляем ответное сообщение клиенту

stream.Write(response, 0, response.Length);

// Закрываем соединение

client.Close();

Console.WriteLine("Соединение закрыто.");

}

}

static bool CheckData(string data)

{

// Здесь можно реализовать различные проверки данных на сервере

// В данном примере просто проверяем, равно ли полученное сообщение "Hello"

return data == "Hello";

}

}

```

Сервер будет слушать входящие подключения на заданном IP-адресе и порту. Когда клиент соединяется с сервером, сервер принимает данные от клиента, проверяет их с помощью функции CheckData() и отправляет ответное сообщение клиенту.

Пример предполагает, что клиент уже отправляет данные на сервер. Вы можете использовать класс TcpClient или другой способ отправки данных на сервер для тестирования кода.

Обратите внимание, что данный пример является базовой реализацией и может быть доработан согласно вашим потребностям.

**31. Создать файл hello\_world.sln в локальной копии репозитория (который должен был появиться). Реализовать программу Hello world на языке C# используя плохой стиль кода. Например, после заголовочных файлов вставьте строку using namespace. «Запуште» изменения в удалёный репозиторий.**

орошо, чтобы создать файл "hello\_world.sln" и реализовать программу "Hello World" на языке C# с использованием плохого стиля кода, выполните следующие шаги:

1. Откройте Visual Studio или любую другую IDE для разработки на языке C#.

2. Создайте новый проект: выберите "File" (Файл) -> "New Project" (Новый проект).

3. В окне "New Project" (Новый проект) выберите "Visual C#" (Визуальный C#) в меню слева и затем выберите "Console App (.NET Framework)" (Консольное приложение (.NET Framework)).

4. Укажите имя проекта, например, "HelloWorldBadStyle" (ПриветМирПлохойСтиль), и выберите местоположение для сохранения проекта.

5. Нажмите кнопку "OK" (ОК) для создания проекта.

6. В файле "Program.cs" разместите следующий код:

using System;

namespace HelloWorldBadStyle

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Hello, world!");

Console.ReadKey();

}

}

}

Обратите внимание, что здесь мы сделали плохой стиль кода, вставив строку "using System;" сразу после объявления пространства имён "namespace". В хорошем стиле кода строка "using System;" должна быть выше объявления пространства имён.

7. Сохраните файл "Program.cs".

8. Запустите программу, чтобы убедиться, что она работает должным образом.

Теперь, чтобы "запушить" изменения в удаленный репозиторий, выполните следующие команды в Git Bash или другом терминале:

1. Перейдите в директорию локальной копии репозитория с помощью команды "cd /путь/к/директории".

2. Выполните команду "git add ." для добавления всех файлов и изменений в индекс Git.

3. Выполните команду "git commit -m 'Add hello\_world.sln and implement Hello World program with bad code style'" для фиксации изменений с комментарием.

4. Наконец, выполните команду "git push origin master" для загрузки изменений в удаленный репозиторий.

Теперь ваши изменения, включая файл "hello\_world.sln" и программу "Hello World" со стилем плохого кода, будут отправлены в удаленный репозиторий.

**32. Изменить исходный код так, чтобы программа через стандартный поток ввода переменных запрашивала имя пользователя. А в стандартный поток вывода выводилось сообщение Hello world from @name, где @name имя пользователя. Провести «коммит» новой версии программы.**

онечно! Вот измененный код для программы "Hello World", которая запрашивает имя пользователя и выводит приветственное сообщение:

using System;

namespace HelloWorld

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Введите ваше имя:");

string name = Console.ReadLine();

Console.WriteLine("Hello world from @" + name);

Console.ReadKey();

}

}

}

Здесь мы добавили строку Console.WriteLine("Введите ваше имя:"); для запроса ввода имени пользователя с помощью команды Console.ReadLine();. Затем мы использовали введенное имя для вывода приветственного сообщения.

Чтобы "закоммитить" новую версию программы, вы можете выполнить следующие команды в Git Bash или другом терминале:

1. Перейдите в директорию локальной копии репозитория с помощью команды cd /путь/к/директории.

2. Выполните команду git add . для добавления изменений в индекс Git.

3. Выполните команду git commit -m 'Modify Hello World program to prompt user for their name and display a customized greeting message' для фиксации изменений с комментарием.

4. Наконец, выполните команду git push origin master для загрузки изменений в удаленный репозиторий.

Теперь ваша программа "Hello World" будет запрашивать имя пользователя и выводить приветственное сообщение с использованием этого имени.

**33. Проверить, что история коммитов доступна в удалённом репозитории.**

Для проверки доступности истории коммитов в удалённом репозитории нужно выполнить следующие шаги:

1. Установите Git, если его ещё нет на вашем компьютере.

2. Откройте командную строку (Windows) или терминал (Mac/Linux).

3. Перейдите в директорию, где находится ваш локальный репозиторий. Для этого используйте команду `cd <путь к директории>`.

4. Проверьте текущий статус вашего локального репозитория с помощью команды `git status`. Она покажет, находитесь ли вы в ветке, которую хотите проверить.

5. Проверьте, какие удалённые репозитории связаны с вашим локальным репозиторием, с помощью команды `git remote -v`. Это покажет вам URL-адреса удалённых репозиториев.

6. Введите команду `git log`. Она покажет вам историю коммитов для текущей ветки вашего локального репозитория.

7. Для просмотра истории коммитов в удалённом репозитории вы можете использовать команду `git log origin/<имя ветки>`. Например, если ветка называется "main", то команда будет выглядеть как `git log origin/main`. Эта команда покажет вам историю коммитов для указанной ветки в удалённом репозитории.

Если видите историю коммитов, значит, она доступна в удалённом репозитории. Если не видите историю коммитов или получаете сообщение об ошибке, возможно, проблема связана с настройкой удалённого репозитория или отсутствием связи с ним.

**34. В локальной копии репозитория создать локальную ветку patch1. Commit, push локальную ветку в удалённый репозиторий. Проверить, что ветка patch1 доступна в удалённом репозитории.**

Для создания и отправки локальной ветки patch1 в удаленный репозиторий, выполните следующие шаги:

1. Откройте командную строку или терминал и перейдите в локальную копию репозитория.

2. Убедитесь, что вы находитесь на главной ветке репозитория (обычно это ветка master или main). Вы можете выполнить команду `git branch` для просмотра текущей ветки и `git status` для проверки состояния репозитория.

3. Создайте локальную ветку patch1 с помощью команды `git branch patch1`. Эта команда создаст ветку patch1 на основе текущей ветки.

4. Перейдите на ветку patch1 с помощью команды `git checkout patch1`.

5. Выполните необходимые изменения и добавьте их в индекс с помощью команды `git add <файлы>` или `git add .` для добавления всех измененных файлов.

6. Создайте коммит, описывающий ваши изменения, с помощью команды `git commit -m "Описание коммита"`.

7. Отправьте локальную ветку patch1 в удаленный репозиторий с помощью команды `git push -u origin patch1`. С помощью опции `-u` вы установите отслеживание удаленной ветки, чтобы в будущем можно было просто использовать команду `git push` для отправки изменений.

8. После запуска команды `git push`, убедитесь, что ветка patch1 доступна в удаленном репозитории. Вы можете проверить это, открыв удаленный репозиторий в веб-браузере и перейдя на страницу веток. Ветка patch1 должна отображаться в списке веток.

**35. В удалённом репозитории выполнить слияние PR patch1 -> master и удалить ветку patch1 в удаленном репозитории. Локально выполнить pull. С помощью команды git log просмотреть историю в локальной версии ветки master.**

Для выполнения этих действий вам понадобятся следующие команды:

1. Перейдите в локальную версию ветки master:

```

git checkout master

```

2. Получите обновления из удалённого репозитория и выполните слияние PR patch1 -> master:

```

git pull origin master

```

3. Удалите ветку patch1 в удаленном репозитории:

```

git push origin --delete patch1

```

4. Выполните pull, чтобы получить изменения удаленного репозитория в ваш локальный репозиторий:

```

git pull origin master

```

5. Используйте команду git log, чтобы просмотреть историю коммитов в локальной версии ветки master:

```

git log

```

Эти команды помогут вам выполнить все указанные действия