БИЛЕТ 1

Репозитории - хранилище информации, связанной с проектом разработки программного продукта в течение всего его жизненного цикла. В современных инструментальных системах репозитории приобретают роль фундамента всей информационной среды

В репозитории хранятся файлы проекта и производятся над ними нужные операции: создаются ветки для разных разработчиков и файлов, производится деление на версии, пишутся комментарии, осуществляется сборка конечного результата в готовый файл проекта. Главная мастер-ветка осуществляет окончательный сбор проекта. Репозиторий состоит из следующих элементов: Централизованные, локальные и распределенные версии Git 1. Локальная система.

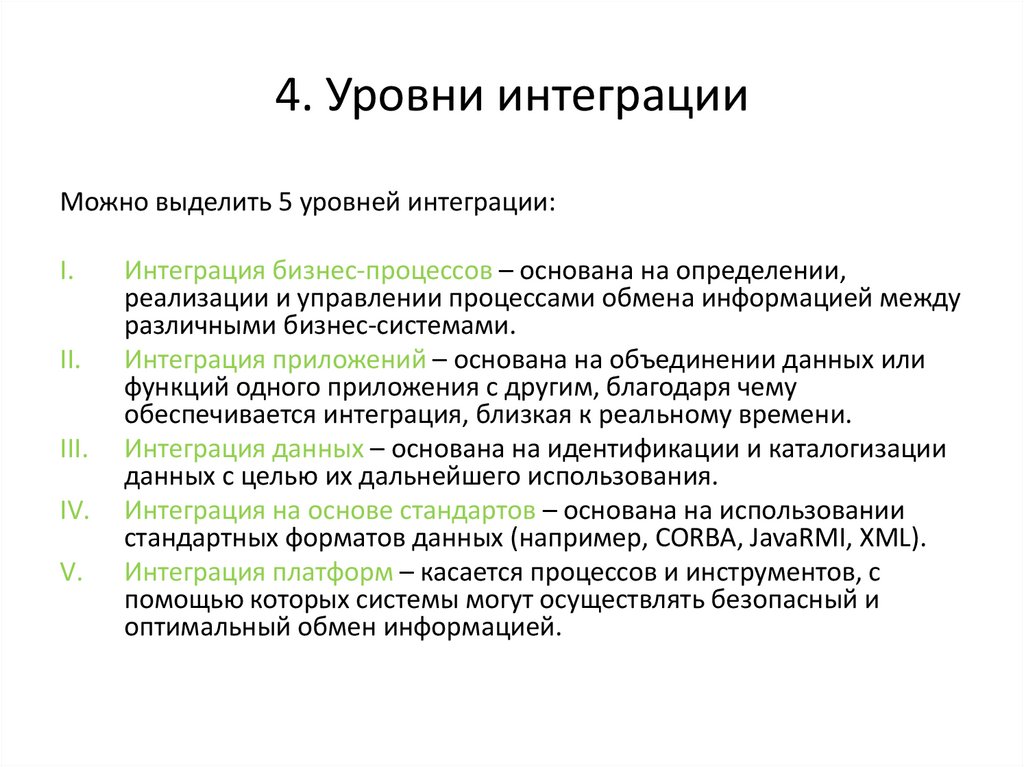
БИЛЕТ 2

Доступны 3 вида интеграции:

* облачная;
* локальная;
* гибридная.

Интеграция – процесс разработки и внедрения программного обеспечения, с помощью которого отдельные компоненты могут быть связаны в единую систему. Такое объединение позволяет поддерживать бизнес-процессы и оперативно обмениваться информацией.

Главная задача процесса – обеспечение безопасного и бесперебойного обмена информацией между программными продуктами, которые изначально не предназначены для совместной работы. Например, программное обеспечение для электронного документооборота между предприятием и его клиентами, организация цепей поставок, ERP-системы, облачные технологии, аналитические модули, системы самообслуживания и т. д.



БИЛЕТ 3

Автоматизация бизнес-процессов (АБП) – это использование информационных технологий для оптимизации и улучшения бизнес-процессов в организации. АБП позволяет автоматизировать различные виды бизнес-процессов, такие как:

1. Производственные процессы – управление производственными линиями, контроль качества продукции, управление запасами и т.д.

2. Логистические процессы – управление поставками, складским хозяйством, транспортировкой товаров и т.д.

3. Финансовые процессы – бухгалтерский учет, управление денежными средствами, налоговое планирование и т.д.

4. Маркетинговые процессы – управление продажами, анализ рынка, управление брендом и т.д.

Основные преимущества автоматизации бизнес-процессов:

1. Увеличение эффективности и производительности бизнес-процессов;

2. Сокращение времени на выполнение задач и уменьшение затрат на их выполнение;

3. Улучшение качества продукции или услуг;

4. Увеличение прозрачности и контроля над бизнес-процессами;

5. Улучшение коммуникации и сотрудничества между отделами и сотрудниками.

Необходимая функциональность системы автоматизации бизнес-процессов включает в себя:

1. Моделирование бизнес-процессов;

2. Управление задачами и проектами;

3. Управление документами и данными;

4. Анализ и отчетность;

5. Интеграция с другими системами;

6. Управление доступом и безопасностью;

7. Масштабируемость и гибкость системы.

БИЛЕТ 4

Выбор источников и приемников данных зависит от конкретной задачи и требований к системе. Например, для автоматизации производственного процесса могут использоваться датчики, считывающие информацию о температуре, давлении, влажности и других параметрах, а также системы управления, которые обрабатывают эти данные и принимают решения. Сопоставление объектов данных также зависит от конкретной задачи. Например, при разработке базы данных для интернет-магазина необходимо определить, какие объекты данных будут храниться в базе данных (товары, категории, заказы, пользователи и т.д.), а также как они будут связаны между собой (например, один заказ может содержать несколько товаров, каждый товар может принадлежать к определенной категории и т.д.). Важно учитывать требования к производительности и безопасности при проектировании базы данных.

БИЛЕТ 5

Контроль версий - это система управления изменениями в коде и документации проекта. Она позволяет отслеживать изменения, вносимые в файлы, сохранять их в истории и возвращаться к предыдущим версиям при необходимости.

Git - одна из самых популярных систем контроля версий. Она работает по принципу распределенной системы, что означает, что каждый участник команды имеет полную копию репозитория и может работать локально, не завися от доступности центрального сервера. Изменения вносятся в локальную копию, а затем отправляются на сервер, где объединяются с изменениями других участников.

Организация работы команды в системе контроля версий включает в себя следующие шаги:

1. Создание репозитория на сервере.

2. Клонирование репозитория на локальные компьютеры участников команды.

3. Создание веток для разных функциональных блоков проекта.

4. Работа над задачами в отдельных ветках.

5. Отправка изменений на сервер и объединение с основной веткой.

6. Решение конфликтов при объединении изменений.

7. Регулярное обновление локальных копий репозитория из центрального репозитория.

Такая организация работы позволяет участникам команды работать параллельно над разными задачами, отслеживать изменения в коде и документации, а также быстро возвращаться к предыдущим версиям при необходимости.

БИЛЕТ 6

Отладка программных продуктов - это процесс нахождения и исправления ошибок в программном коде. Отладка требует специальных средств и тестирования, чтобы выявить ошибки.

Средства отладки (debugger) - это программы или инструменты, которые используются для отслеживания ошибок в программном коде. Они позволяют разработчикам проверять, как код работает в режиме выполнения, выявлять возможные ошибки и исправлять их.

Средства отладки могут предоставлять информацию об ошибке, а также помочь в процессе исправления ошибок. Они могут использоваться для многих языков программирования, таких как Java, C++, C#, Python и другие. Некоторые примеры средств отладки включают Visual Studio Debugger, GDB, PyCharm Debugger, Xdebug и многие другие.

БИЛЕТ 7

Инструменты отладки – это программные приложения, которые используются для нахождения и исправления ошибок в коде программы. Они помогают программисту выявлять и устранять ошибки, улучшать производительность и качество программы.

Виды инструментов отладки:

IDE (интегрированная среда разработки). Они включают в себя отладчики, которые могут обнаруживать ошибки, позволяют установить точки останова, пошагово отслеживать выполнение кода и т.д. Примеры IDE: PyCharm, Visual Studio, Eclipse, NetBeans.

Программы для профилирования. Эти программы отслеживают производительность программы и находят проблемы с производительностью. Примеры программ для профилирования: JetBrains dotTrace, Visual Studio Profiler, Intel VTune Amplifier.

Инструменты для динамического анализа кода. Эти инструменты обнаруживают ошибки и сбои в исполнении программы в реальном времени. Примеры инструментов для динамического анализа кода: Valgrind, Purify, Insure++.

Инструменты для статического анализа. Они позволяют анализировать код программы на предмет возможных ошибок без ее запуска. Примеры инструментов для статического анализа: SonarQube, Checkstyle, PyLint.

Применение инструментов отладки:

Обнаружение и исправление ошибок в коде;

Анализ производительности программы;

Устранение утечек памяти;

Тестирование кода на совместимость и безопасность;

Улучшение качества программного продукта;

Оптимизация работы программы.

БИЛЕТ 8

Класс Debug в .NET Framework является вспомогательным инструментом для отладки приложений. Этот класс содержит свойства, методы, сообщения и события, которые могут быть использованы для создания отладочных сообщений, вывода информации об исключениях, установки точек останова и т.д.

Свойства класса Debug:

AutoFlush - позволяет управлять автоматической очисткой буфера вывода, когда он переполняется.

IndentLevel - определяет уровень отступа для метода отладки Indent.

IndentSize - устанавливает размер отступа для метода отладки Indent.

Listeners - определяет список объектов, используемых классом Debug для записи отладочной информации.

Методы класса Debug:

Assert - прерывает выполнение программы, если результат выражения является ложным.

Write - выводит заданное значение в строку отладки.

WriteLine - выводит заданное значение в строку отладки, дополненную символом новой строки. Indent - увеличивает уровень отступа.

Unindent - уменьшает уровень отступа.

Примеры использования класса Debug:

Вывод отладочной информации в консоль:

Debug.Write("Начало работы программы"); Debug.WriteLine("Чтение файла данных...");

Установка точки останова для отладки: Debug.Assert(value == 10, "Значение переменной не равно 10");

Применение отступов для вывода информации в консоль: Debug.IndentLevel = 1; Debug.IndentSize = 4; Debug.WriteLine("Изменена конфигурация приложения");

Использование объекта DebugListener для записи отладочной информации в файл: FileStream logFile = new FileStream("log.txt", FileMode.Create); TextWriterTraceListener listener = new TextWriterTraceListener(logFile); Debug.Listeners.Add(listener); Debug.WriteLine("Ошибка базы данных: " + ex.Message);

БИЛЕТ 9

Ручное тестирование – это процесс проверки функциональности приложения путем взаимодействия с ним вручную. Тестировщик создает тест-кейсы и выполняет их вручную, чтобы проверить правильность работы приложения. Ручное тестирование может быть эффективным в случае малого количества тест-кейсов, однако в случае большого количества тестов это занимает много времени и требует большой команды тестировщиков.

Автоматизированное тестирование – это процесс тестирования, в котором используются инструменты и сценарии для автоматического выполнения тестов. Тестовые скрипты создаются вручную или генерируются автоматически с помощью инструментов автоматизации тестирования. Автоматизированное тестирование может быть более эффективным и быстрым, так как многие тесты могут быть выполнены параллельно. Однако, для создания тестовых скриптов требуется некоторое количество времени и усилий.

В целом, ручное и автоматизированное тестирование работают вместе для обеспечения качества разрабатываемого программного обеспечения. Ручное тестирование может быть использовано для проверки новых функций и ручной проверки важных сценариев использования. Автоматизированное тестирование может быть использовано для быстрого выполнения повторяемых тестов и вызовов функций API.

Билет 10

TCP (Transmission Control Protocol) и UDP (User Datagram Protocol) являются двумя основными транспортными протоколами, используемыми в сетях. Они оба служат для установления соединения между компьютерами и обмена данными между ними, но различаются в способе доставки этих данных.

TCP является протоколом, который гарантирует правильность и последовательность передачи данных, что делает его более надежным, но медленным. Он обеспечивает установление соединения между компьютерами, передачу данных в очередности, проверку ошибок и повторную передачу данных при возникновении ошибок. Протокол TCP используется в приложениях, где требуется надежность доставки данных, таких как электронная почта, файловые передачи и доступ к веб-сайтам.

UDP, с другой стороны, не гарантирует правильность и последовательность передачи данных, что делает его быстрее, но менее надежным. Он не устанавливает соединения между компьютерами и просто отправляет данные, как они приходят, без проверки на ошибки и без повторной передачи данных. Протокол UDP используется в быстрых приложениях, таких как потоковое видео и онлайн-игры, где требуется быстродействие и низкая задержка.

Сходства между TCP и UDP заключаются в том, что они оба являются транспортными протоколами, обеспечивают передачу данных между компьютерами и используют IP-адреса и порты для идентификации устройств, между которыми передаются данные.

Различия между TCP и UDP заключаются в надежности доставки данных, скорости передачи данных, тональности и способе установления соединения между компьютерами. TCP обеспечивает надежную доставку данных, но медленнее из-за проверки ошибок и повторной передачи данных, а UDP обеспечивает быструю передачу данных, но менее надежно без проверки ошибок и повторной передачи данных. TCP устанавливает соединение между компьютерами, а UDP отправляет данные без установления соединения.

Билет 11

Стандарты форматирования сообщений: организация данных. Стандарты форматирования сообщений предусматривают определенные правила организации данных для обеспечения удобства чтения и понимания информации. Вот некоторые из них:

1.1. Оформление заголовка сообщения. Заголовок должен содержать краткую суть сообщения, чтобы получатель сразу понял, с чем связано его содержание. Желательно указать тему или проблему, которой посвящено сообщение.

1.2. Подробное изложение информации. Сообщение должно содержать подробное описание событий, фактов или просьбы. Необходимо ограничивать себя одной темой, чтобы не создавать путаницу у получателя.

1.3. Перечисление элементов. Если в сообщении есть список элементов, они должны быть указаны в виде нумерованного или маркированного списка.

1.4. Использование абзацев. В тексте сообщения необходимо использовать абзацы для разделения логически связанных частей информации. Такой подход облегчает чтение сообщения и понимание его содержания.

1.5. Использование выделения. В сообщении можно использовать выделение текста (жирный шрифт, курсив, подчеркивание), чтобы подчеркнуть ключевые слова и фразы.

1.6. Оформление подписи. В конце сообщения следует указать подпись, которая содержит информацию об отправителе (имя, должность, контактная информация), чтобы получатель мог обратиться к нему по необходимости.

Билет 12

Методы и средства организации тестирования могут быть разнообразными, но главная задача при этом – обеспечить максимальную эффективность и достоверность результата тестирования. Наиболее распространенными методиками тестирования являются:

Ручное тестирование – проводится вручную с помощью специальных тестовых сценариев, которые заранее разработаны тестировщиком.

Автоматизированное тестирование – проводится с использованием специальных программных комплексов, предназначенных для автоматического выполнения тестовых сценариев.

Стресс-тестирование – проводится для определения максимальной нагрузки на приложение или веб-сайт. Для этого создается условие, когда на систему накладывается большое количество запросов, превышающее ее возможности.

Пользовательское тестирование – проводится с помощью реальных пользователей, которые проверяют функциональность приложения или веб-сайта.

Тестирование на готовность – проводится перед выпуском приложения или веб-сайта на базовое качество и готовность к выходу на рынок.

Эксплуатационное тестирование – проводится после выпуска приложения или веб-сайта на рынок для проверки его работоспособности в реальных условиях и выявления багов.

Регрессионное тестирование – проводится после внесения изменений или улучшений в систему, чтобы убедиться, что изменения не повлияли на следующую работу системы.

Оценка безопасности – проводится для оценки уровня защиты приложения или веб-сайта от злоумышленников и хакеров.

Для проведения тестирования существует множество средств и инструментов, которые могут значительно упростить процесс тестирования и повысить качество конечного продукта. К ним относятся различные средства автоматизации тестирования, тестовые фреймворки, утилиты для управления тестами и множество других программных продуктов.

БИЛЕТ 13

Существует множество инструментов и методик для анализа качества программных продуктов в среде разработки. Некоторые из них включают:

1.1. Статический анализ кода: - Анализаторы кода: проверка кода на соответствие установленным стандартам и соглашениям; - Поиск ошибок: инструменты, которые автоматически находят ошибки в коде; - Анализаторы зависимостей: помогает обнаружить зависимости между различными компонентами программы;

1.2. Динамический анализ кода: - Модульное тестирование: тестирование кода единицами (методами или функциями); - Интеграционное тестирование: тестирование взаимодействий между компонентами программы; - Нагрузочное тестирование: тестирование производительности программы при работе с большим количеством данных или пользователей;

1.3. Анализ пользовательского опыта: - Анализ пользовательских отзывов: обзоры пользователей на маркетплейсах или форумах; - Анализ метрик: изучение поведения пользователей в программе с помощью метрик, таких как время работы в программе или количество ошибок.

Целью использования таких инструментов является повышение качества программного продукта, увеличение производительности и улучшение взаимодействия с пользователями.

БИЛЕТ 14

Исключительные ситуации (exceptions) – это ошибки в работе программы, которые возникают, когда программа не может выполнить какую-то задачу по какой-то причине. Исключения могут быть вызваны ошибками ввода-вывода, недостаточным объемом памяти, отсутствием файлов и другими причинами.

Существует несколько видов исключений:

RuntimeExceptions, которые генерируются во время выполнения программы. Например: ArithmeticException, ArrayIndexOutOfBoundsException, NullPointerException и т.д.

Checked exceptions, которые должны быть обработаны программистом, использующим метод, который может вызвать такое исключение. Например: IOException, FileNotFoundException, ClassNotFoundException и т.д.

Error, которые обычно свидетельствуют о том, что что-то пошло не так с самой программой. Например: OutOfMemoryError, StackOverflowError и т.д.

Также есть возможность использовать блок finally для выполнения каких-то действий независимо от того, было ли сгенерировано исключение или нет.

try {

// выполнение какого-то действия

} catch (Exception e)

{

// обработка исключения

} finally

{

// выполнение действий независимо от того, было ли исключение или нет

}

БИЛЕТ 15

Методы и способы идентификации сбоев и ошибок используются для выявления ошибок, возникающих в процессе работы программного или аппаратного обеспечения. Ошибки могут возникать в различных местах, начиная от аппаратных компонентов, продвигаясь через операционную систему и заканчивая приложениями.

Типы ошибок:

Ошибки синтаксиса – возникают, когда программа не соответствует правилам синтаксиса языка программирования.

Ошибки логики – возникают, когда программа содержит ошибки в алгоритмах, что приводит к неверным результатам.

Ошибки времени выполнения – возникают во время выполнения программы, когда происходят непредвиденные ситуации.

Ошибки ввода-вывода – связанные с взаимодействием программного обеспечения и устройств ввода-вывода.

Действия для выявления ошибок:

Тестирование – проведение тестовых испытаний программного обеспечения.

Мониторинг системы – отслеживание работы системы и ее компонент.

Логирование – сохранение информации о работе программы, включая ошибки и сбои.

Анализ кода – изучение и анализ исходного кода программы.

Отладка – проверка программы в ходе ее выполнения в режиме отладки.

Использование дополнительных инструментов – например, средств анализа производительности или инструментов мониторинга состояния процессов.

БИЛЕТ 16

Существует несколько методов выявления ошибок системных компонентов, каждый из которых имеет свои преимущества и недостатки:

Тестирование. Это самый распространенный метод выявления ошибок в системных компонентах. Он заключается в создании тестовых случаев и проверке корректности работы компонентов при их выполнении. Метод часто используется как первичный этап тестирования, но требует больших затрат времени и ресурсов.

Статический анализ кода. Этот метод является более быстрым и эффективным, чем тестирование, и позволяет выявлять ошибки до их запуска в работу системы. Анализаторы кода могут автоматически проверять правильность использования функций, переменных и операторов, а также обнаруживать подозрительный код.

Мониторинг производительности. Компоненты системы могут работать неэффективно, что приводит к ошибкам и снижению производительности. Модули мониторинга производительности могут выявить проблемы и предоставить данные для их устранения.

Анализ журналов. Журналы ошибок могут содержать полезную информацию о возникающих проблемах в системе. Анализ этих данных может помочь выявить типичные ошибки и устранить их причины.

Использование инструментов диагностики. Существует множество инструментов диагностики, которые помогают выявлять ошибки в системе, включая инструменты для анализа производительности, трассировки кода и мониторинга состояния сетевых соединений.

В целом, правильное использование всех этих методов может значительно повысить качество системных компонентов и уменьшить риск возникновения ошибок.

БИЛЕТ 17

Язык UML (Unified Modeling Language) - это стандарт для визуального моделирования систем. Он обладает рядом преимуществ и недостатков.

Достоинства языка UML:

Возможность использования на всех этапах жизненного цикла проекта, начиная от анализа и проектирования, и заканчивая тестированием и технической документацией.

Универсальность - многие методы разработки ПО используют UML в качестве языка моделирования.

Гибкость и расширяемость - возможность переопределить и расширить стандартные элементы языка.

Язык обеспечивает удобство использования при работе в команде.

Возможность автоматической генерации кода на основе моделей.

Недостатки языка UML:

Моделирование требует умения использовать язык, что может быть трудно для непрофессионалов.

Необходимость постоянного обучения и совершенствования навыков в использовании UML.

Не всегда возможно использовать UML полностью для моделирования конкретной системы.

Часто возникает проблема несоответствия модели на UML с актуальным состоянием системы в реальности.

Виды диаграмм:

Диаграмма классов - позволяет описать все классы системы, связи между ними, свойства и методы классов, их наследование и ассоциации.

Диаграмма действий - описывает детальный алгоритм работы системы.

Диаграмма состояний - позволяет отобразить различные состояния объекта в системе и переходы между ними.

Диаграмма последовательностей - описывает порядок выполнения действий и взаимодействие между объектами в системе.

Диаграмма развертывания - показывает аппаратное и программное обеспечение, на котором будет работать система и связи между ними.

Диаграмма компонентов - представляет различные компоненты системы и связи между ними.

Диаграмма случаев использования - описывает основные случаи использования системы и взаимодействие между пользователями и системой.

БИЛЕТ 18

Система будет использоваться в различных сценариях, и это позволяет разработчикам более эффективно понимать требования к системе и разрабатывать более эффективные решения. Все это делает диаграмму важным инструментом для успешной разработки ПО.

UML- диаграмма классов: элементы диаграммы, взаимосвязи.

UML-диаграмма классов представляет собой графическое изображение классов и их взаимосвязей в системе. Она состоит из следующих элементов:

Классы – прямоугольники с названием класса внутри, содержащие атрибуты (переменные класса) и методы (функции класса).

Ассоциации – линии между классами, обозначающие связи между объектами.

Агрегация – знак алмаза, указывающий, что объекты в составе другого объекта являются его частями.

Композиция – композиция – знак алмаза, где объекты в составе другого объекта имеют жесткую связь и удаление главного объекта ведет к удалению внутренних объектов.

Наследование – стрелка от наследника к базовому классу, указывающая, что наследник получает все методы и атрибуты базового класса.

Интерфейсы – овалы с названием интерфейса внутри, используемые для описания методов, которые должны быть реализованы классами, реализующими этот интерфейс.

Взаимосвязи между элементами диаграммы могут выражаться следующими способами:

Ассоциация – связь между двумя классами.

Агрегация – связь между классом и его частью.

Композиция – связь между классом и его внутренними частями.

Наследование – связь между базовым классом и его наследниками.

Реализация – связь между классом и интерфейсом, который он реализует.

БИЛЕТ 19

UML-диаграмма вариантов использования (Use Case Diagram) - это диаграмма, которая показывает, какие действия и операции могут выполнять пользователи (актеры) с использованием системы.

Назначение UML-диаграммы вариантов использования заключается в том, чтобы проиллюстрировать функциональность системы и ее взаимодействие с внешним окружением. Это помогает проектировщикам и разработчикам определить функциональность системы и ее взаимосвязи.

Элементы UML-диаграммы вариантов использования:

1. Актеры - это люди, системы или другие взаимодействующие объекты, которые используют систему;
2. Варианты использования - это функциональные возможности или действия, которые могут быть выполнены актерами с помощью системы;
3. Отношения между актерами и вариантами использования - это связи, которые определяют, какие актеры могут выполнять какие варианты использования. Например, актер может быть связан с одним или несколькими вариантами использования, а один вариант использования может быть связан с несколькими актерами.

UML-диаграмма вариантов использования дает возможность визуализировать, как система будет использоваться в различных сценариях, и это позволяет разработчикам более эффективно понимать требования к системе и разрабатывать более эффективные решения. Все это делает диаграмму важным инструментом для успешной разработки ПО.

БИЛЕТ 20

Основные этапы документирования результатов тестирования:

Планирование тестирования: определение целей и задач, выбор методов тестирования, составление плана тестирования.

Написание тест-кейсов: создание подробного описания тестовых сценариев, выбор критериев оценки результатов тестирования.

Проведение тестирования: осуществление запланированных тестовых сценариев, фиксация результатов.

Анализ результатов: анализ полученных данных, выявление ошибок и дефектов, определение причин их возникновения.

Документирование результатов: фиксация всех результатов тестирования, включая найденные ошибки и дефекты, описание условий их возникновения, рекомендации по устранению их последствий.

Подготовка отчета: составление подробного отчета о проделанной работе, включающего описание всех этапов тестирования, результаты и анализ результатов, рекомендации по улучшению продукта.

Представление отчета руководству: передача отчета верхнему руководству для анализа и принятия решений о дальнейшем улучшении продукта.

Начало формы