1. **Понятие ревьюирование. Цели и задачи ревьюирование.**

Ревьюирование – это структурированный процесс оценки работы, документации или кода, направленный на выявление ошибок и недостатков, улучшения качества и повышение эффективности.

Цели:

* Повышение качества
* Обмен знаниями
* Улучшение процессов
* Соответствие стандартам
* Проверка безопасности
* Снижение затрат

Задачи:

* Анализ кода/документации
* Оценка производительности
* Проверка безопасности
* Обратная связь

Существует четыре способа произвести код-ревью:

1. Ревью "Из-за Плеча" – такие проверки выполняются за рабочей станцией разработчика, где опытный участник команды просматривает новый код, выдвигая свои предложения в процессе обсуждения. Это самый простой способ проведения код-ревью, который не требует определенной структуры.
2. Почтовая Рассылка – географически удаленные команды по традиции опирались на электронную почту для ревью кода. Раньше почта была основным способом коммуникации ввиду ее универсальности. Организации с открытым кодом часто поддерживали публичный список рассылки, который также служил средством для обсуждения и предоставления обратной связи по коду.
3. Парное программирование – это непрерывный процесс ревью кода. Двое разработчиков находятся за рабочей станцией, но лишь один из них активно кодит, в то время как другой предоставляет обратную связь в реальном времени.
4. С Помощью Инструментов – такой вид код-ревью включает в себя использование специализированного программного обеспечения (ПО) в целях облегчения процесса ревью кода.
5. **Этапы и объекты ревьюирования.**

Этапы:

* Инициализация
* Планирование
* Подготовка (экспертиза)
* Обсуждение
* Завершение

Объекты:

* Исходный код
* Документация
* Проектные артефакты
* Процессы

1. **Цели, корректность и направления анализа программных продуктов.**

Цели:

* разработку технических инструкций;
* преобразование требований в форму, понятную разработчику, то есть определение подсистем, компонентов и классов.

Корректность анализа заключается в проверке корректности программ, для чего используются эталоны и методы проверки. Например, исследуется программный код на предмет ошибок и отклонения от алгоритма.

Направления:

* Определение потенциальной архитектуры. Определяется первоначальный набор архитектурно значимых элементов и механизмов реализации, структура системы.
* Уточнение архитектуры. Включает определение механизмов проектирования, элементов проекта, объединение существующих элементов проекта.
* Анализ поведения. Цель — преобразовать описания поведения в набор элементов проекта (классы, отношения, операции и др.).

1. **Примеры сравнительного анализа программных продуктов**

Сравнительный анализ программных продуктов — это процесс сравнения особенностей и характеристик различных программных средств с целью выявления их качества, безопасности и производительности.

Цели:

* Выявление особенностей и характеристик различных программных продуктов.
* Оценка качества, безопасности и производительности программных продуктов.

Шаги:

* Формирование комиссии экспертов.
* Определение методики оценки для каждого критерия.
* Ранжирование значений интегральных оценок для каждого продукта по убыванию.

Сравнительный анализ программных продуктов для автоматизации бухгалтерского учёта. сравниваются программы «1С», «Парус» и «Инфо-Бухгалтер». Указывается, что «1С» считается наиболее распространённой, функциональной и доступной для бухгалтера программой.

Сравнительный анализ программного обеспечения для создания веб-сайтов. сравниваются программы AdobeDreamweaver, Brackets и VisualStudioCode. У каждой из них есть свои достоинства и недостатки, связанные с техническими требованиями, функционалом, стоимостью и т. д..

1. **Цели, задачи и методы исследования программного кода**

Цели и задачи:

* Изучение и анализ качества кода
* Поиск и устранение ошибок и дефектов
* Улучшение читаемости и поддержаемости кода
* Проверка соответствия стандартам и требованиям
* Оптимизация производительности

Методы:

* Статический анализ
* Динамический анализ
* Тестирование кода

1. **Механизмы и контроль внесения изменений в код**

**Для механизмов и контроля внесения изменений в код используются системы контроля версий**.  Они позволяют нескольким разработчикам работать над одной кодовой базой одновременно без конфликтов, обеспечивая централизованный или распределённый репозиторий, где хранятся и отслеживаются все изменения кода.

**Некоторые возможности систем контроля версий:**

* **Хранение нескольких версий**
* **Фиксация, кто и когда сделал изменение**.
* **Объединение изменений**.
* **Откат к предыдущим версиям**.

Также для контроля внесения изменений в код может использоваться **база данных ошибок**. В ней хранится информация об обнаруженных ошибках и приоритетах их исправления. Для каждой ошибки назначают ответственных за её исправление, обозначают срок исправления. После внесения исправлений об этом делается отметка и сохраняется код с исправленной ошибкой.

1. **Обратное проектирование.**

Обратное проектирование – это процесс изучения и анализа уже существующего продукта с целью понимания его структуры, работы и принципов функционирования.

Цели:

* Восстановление утраченной документации
* Анализ конкуренции
* Совместимость и устранение уязвимостей
* Безопасность и устранение уязвимостей

Этапы:

* Сбор информации
* Моделирование
* Анализ и документирование
* Реконструкция

1. **Анализ потоков данных.**

Анализ потоков данных – это метод анализа программы или системы, который рассматривает передачу данных между различными частями программы или подсистемами.

Цели:

* Обнаружение ошибок
* Оптимизация производительности
* Анализ безопасности
* Управление данными

Методы:

* Статический метод
* Динамический метод

1. **Понятие дизассемблирования.**

Дизассемблирование  - это получение из исполняемого кода программы код на языке ассемблера.

Дизассемблер - программа, осуществляющая дизассемблирование.

Дизассемблер используют для анализа программы, исходный текст которой неизвестен — с целью копирования или взлома.

Так же для поиска ошибок в программах, а также для анализа и оптимизации создаваемого машинного кода.

1. **Понятие программного продукта**

Программный продукт - комплекс взаимосвязанных программ для решения определенной проблемы (задачи) массового спроса, подготовленный к реализации как любой вид промышленной продукции.

Программные продукты могут быть различного типа:

1. Коммерческие.

2. Свободные.

3. Пользовательские.

4. Системные.

5. Прикладные.

1. **Предпроцессинг кода. Интеграция в IDE**

Предпроцессинг кода — это процесс предварительной обработки кода, который включает в себя анализ его структуры и выявление возможных ошибок.

Интегрированная среда разработки (IDE) – это ПО, которое предоставляет разработкам все необходимые инструменты для написания, тестирования, и отладки кода в одном месте.

Например: Visual Studio, Pycharm

IDE играет ключевую роль в современном программировании, предоставляя мощные инструменты для работы с кодом. Они помогают не только писать код, но и управлять проектами, отслеживать изменения и работать в команде.

1. **Валидация кода на стороне сервера и разработчика**

Валидация – установление соответствия между функциями программы и ее целевым назначением.

Валидация кода может выполняться на стороне сервера и на стороне клиента.

Валидация на стороне сервера происходит после отправки формы. Такая проверка надёжнее, так как не зависит от клиента и помогает точнее обнаружить ошибки в данных.

Валидация на стороне клиента – это проверка данных до отправки формы. Она помогает пользователю быстро узнать об ошибках при заполнении полей и исправить их.

Таким образом, за валидацию отвечают и фронтендеры, и бэкендеры. Первые должны настроить валидацию на стороне клиента, вторые – позаботиться о проверке на стороне сервера.

1. **Совместимость и использование инструментов ревьюироваия в различных системах контроля версий**

1. Системы контроля версий: Это программы, которые помогают отслеживать изменения в коде и управлять разными версиями проекта.

2. Инструменты ревьюирования: Это специальные программы или функции, которые позволяют разработчикам проверять и обсуждать изменения в коде перед их добавлением в основную ветку. Примеры: GitHub Pull Requests, GitLab Merge Requests, Gerrit.

3. Совместимость: Многие инструменты ревьюирования интегрируются с популярными СКВ. Например, GitHub и GitLab работают непосредственно с Git. Это позволяет легко создавать запросы на ревью и отслеживать обсуждения.

4. Использование: Обычно процесс выглядит так:

• Разработчик вносит изменения в код.

• Другие разработчики получают уведомление и могут просмотреть изменения.

• Они могут комментировать, задавать вопросы или предлагать улучшения.

• После обсуждения и внесения правок изменения могут быть объединены с основной веткой.

1. **Особенности ревьюирования в Linux. Настройки доступа**

Ревьюирование кода в Linux, как и в других проектах с открытым исходным кодом, имеет свои особенности и практики. Вот некоторые из них:

* Открытость и доступность
* Строгие стандарты кодирования
* Обсуждение и обратная связь
* Фокус на производительности и безопасности

Настройки доступа:

• В проектах с открытым исходным кодом доступ к коду обычно открыт для всех.

• Однако, чтобы внести изменения в основной репозиторий, необходимо получить одобрение.

• Это помогает поддерживать качество кода и предотвратить ошибки.

1. **Типовые инструменты и методы анализа программных проектов**

1. Код-ревью: это структурированный процесс оценки работы, документации или кода, направленный на выявление ошибок и недостатков, улучшения качества и повышение эффективности.

2. Статический анализ: Проводится без выполнения программы и предполагает проверку исходного кода на соответствие определённым правилам и стандартам.

3. Динамический анализ: Проводится во время выполнения программы и позволяет выявить ошибки, которые невозможно обнаружить при статическом анализе.

4. Тестирование: Процесс проверки программного обеспечения на соответствие требованиям и выявление ошибок.

5. Моделирование: Создание абстрактных представлений системы (например, диаграмм), чтобы лучше понять её структуру и поведение.

6. Метрики: мера количественной оценки качества ПО по заданному критерию, система или способ измерений качества ПО.

7. Отчёты об ошибках: Документы, в которых описаны найденные ошибки, их причины и способы их исправления.

1. **Инструментарий различных сред разработки**

**Среда разработки (IDE)** —это ПО, которое предоставляет разработкам все необходимые инструменты для написания, тестирования, и отладки кода в одном месте. Например: Visual Studio, Pycharm.

IDE играет ключевую роль в современном программировании, предоставляя мощные инструменты для работы с кодом. Они помогают не только писать код, но и управлять проектами, отслеживать изменения и работать в команде.

1. Редакторы кода: Легкие программы для написания кода без множества дополнительных функций. Примеры: Visual Studio Code.

2. Системы контроля версий: Это программы, которые помогают отслеживать изменения в коде и управлять разными версиями проекта. Пример: Git.

3. Компиляторы: Программы, которые превращают написанный код на высокоуровневом языке (например, C++) в машинный код, который может выполнять компьютер.

4. Отладчики: программный инструмент, позволяющий программисту наблюдать за выполнением исследуемой программы, останавливать и перезапускать её, прогонять в замедленном темпе, изменять значения в памяти и даже, в некоторых случаях, возвращать назад по времени. Примеры: встроенные отладчики в IDE.

5. Фреймворки: Наборы готовых библиотек и инструментов, которые упрощают разработку приложений.

1. **Инструментарий Java Development Kit**

Java Development Kit (JDK) — это набор инструментов, необходимых для разработки программ на языке Java. Вот его основные компоненты простыми словами:

1. Компилятор (javac): Программа, которая преобразует ваш код на Java в байт-код, который может выполняться виртуальной машиной Java (JVM). Это первый шаг в создании приложения.

2. Java Runtime Environment (JRE): Среда выполнения, которая позволяет запускать Java-программы. Она включает в себя виртуальную машину Java и стандартные библиотеки.

3. Инструменты командной строки: Набор утилит для выполнения различных задач, таких как: java: запускает Java-программы. javadoc: генерирует документацию из ваших комментариев в коде. jar: упаковывает ваши классы и ресурсы в один файл (JAR), чтобы их было легче распространять.

4. Библиотеки стандартных классов: Готовые функции и классы, которые вы можете использовать в своих приложениях, например, для работы с файлами, сетями или графикой.

5. Отладчик (jdb): Инструмент для поиска и исправления ошибок в вашем коде.

1. **Инструментарий NetBeans и другие**

NetBeans — это интегрированная среда разработки (IDE) для программирования на Java и других языках. Вот основные особенности и инструменты, которые она предлагает:

1. Редактор кода: Легкие программы для написания кода без множества дополнительных функций. Примеры: Visual Studio Code.

2. Отладка: это поиск, анализ и устранение ошибок в программном обеспечении, которые были найдены во время тестирования.

3. Управление проектами: Упрощенное создание и управление проектами, включая структуру папок и зависимостей.

4. Интеграция с системами контроля версий: Поддержка Git и других систем для управления версиями вашего кода.

5. Поддержка различных технологий: Возможность работы с веб-технологиями (HTML, CSS, JavaScript).

Кроме NetBeans, существуют и другие популярные IDE для Java:

• Eclipse: Мощная и гибкая среда с большим количеством плагинов и поддержкой различных языков.

• IntelliJ IDEA: Известна своей интеллектуальной подсветкой кода и множеством встроенных инструментов для повышения продуктивности.

1. **Измерительные методы оценки программ: назначение, условия применения.**

Измерительные методы оценки программ основаны на получении информации с использованием инструментальных средств.

Методы:

* Регистрационный метод. Используется во время испытаний или функционирования программного обеспечения, когда регистрируют или подсчитывают определённые события (время и число сбоев или отказов, время передачи управления другим модулям, время начала и окончания работы).
* Органолептический метод. Основан на использовании информации, полученной в результате анализа восприятия органов чувств (зрения, слуха) для определения показателей удобства применения.
* Расчётный метод. Базируется на статистических данных, собранных при проведении испытаний, эксплуатации и сопровождении программного обеспечения. При помощи расчётного метода определяют длительность вычислений, время реакции, показатели надёжности, необходимые ресурсы.
* Экспертный метод. Осуществляется группой экспертов — специалистов, компетентных в решении данной задачи или типа программного обеспечения. Их оценка базируется на опыте и интуиции, а не на непосредственных результатах расчётов или экспериментов.

Назначение измерительных методов оценки программ — получение информации о свойствах и характеристиках программного обеспечения с использованием инструментальных средств.

Условия применения этого метода включают: калибровку метрик на ранних этапах проекта. С его помощью определяют объём программного обеспечения, число исполненных операторов, ветвей в программе, точек входа (выхода), время выполнения ветви программы, время реакции и другие показатели.

1. **Понятие Корректности программ.**

Корректность программ- соответствие программы некоторому эталону или совокупности эталонных правил; отсутствие ошибок в программе по отношению к цели ее разработки.

Корректность является статическим свойством программы, поскольку она не зависит от времени и отражает специфику ошибок разработки программ.

Наиболее полным эталоном корректности программ является программная спецификация.

1. **Эталоны и методы проверки корректности**

|  |  |
| --- | --- |
| Средства проверки корректности | |
| Эталоны | Методы |
| 1) Формализованные правила:  - описания алгоритмов  - описания данных  - описания модулей  - описания комплексов | 1) - синтаксический контроль  - семантический контроль  - структурный контроль |
| 2) Программные спецификации  - модулей  - данных  - взаимодействия модулей | 2)  - валидация  - верификация |
| 3) Тесты  - статические  - динамические | 3) - просмотр / инспекция кода  - структурный и  функциональный контроль |

1. **Метрики, направления применения метрик.**

Метрика - мера количественной оценки качества ПО по заданному критерию, система или способ измерений качества ПО.

Метрика содержит один или несколько оценочных элементов.

Основные направления применения метрик:

* ﻿﻿оценки топологической и информационной сложности программ;
* ﻿﻿оценки надежности программных систем, позволяющие прогнозировать отказовые ситуации;
* ﻿﻿оценки производительности ПО и повышения его эффективности путем выявления ошибок проектирования;
* ﻿﻿оценки уровня языковых средств и их применения;
* ﻿﻿оценки трудности восприятия и понимания программных текстов;
* ﻿﻿оценки производительности труда.

1. **Метрики сложности. Метрики стилистики**

Метрика - мера количественной оценки качества ПО по заданному критерию, система или способ измерений качества ПО.

Метрика содержит один или несколько оценочных элементов

Метрики сложности программ:

* Метрики размера программ – наиболее просты и широко распространены. Традиционной характеристикой размера программ является количество строк исходного текста. К ним относится метрика Холстеда.
* Метрики сложности потока управления программ – с помощью этих оценок оперируют либо плотностью управляющих переходов внутри программ, либо взаимосвязями этих переходов. К ним относятся:
* Метрика Майерса
* Метрики сложности потока данных программ – метрики сложности потока данных, т.е. использования, конфигурации и размещения данных в программах. К ним относятся:
* Метрика спена

Некоторые метрики стилистики:

* Метрика уровня комментированности программ.

Наиболее простой метрикой стилистики и понятности программ является оценка уровня комментированности программы F:

Метрика F отражает насыщенность программы комментариями.

Принято считать, что на каждые десять строк программы должен приходиться минимум один комментарий.

Комментарии распределяются по тексту программы неравномерно.

* Метрики Холстеда
* Метрика изменения длины программной документации

Чем меньше изменений и корректировок вносится в программную документацию, тем более четко были сформулированы решаемые задачи на всех этапах работ.

1. **Исследование программного кода на предмет ошибок и отклонения от алгоритма**

Исследование программного кода на предмет ошибок и отклонения от алгоритма включает в себя два основных типа анализа:

* Статический анализ. Проводится без выполнения программы и предполагает проверку исходного кода на соответствие определённым правилам и стандартам. Это может включать проверку на наличие синтаксических ошибок, соответствие стилю кодирования, а также выявление потенциальных уязвимостей и проблем с производительностью. Примеры инструментов для статического анализа: SonarQube, ESLint, Pylint.
* Динамический анализ. Проводится во время выполнения программы и позволяет выявить ошибки, которые невозможно обнаружить при статическом анализе. Например, Valgrind помогает выявлять утечки памяти, ошибки доступа к памяти и другие проблемы.

Также для исследования бинарного кода на ошибки и корректность работы могут использоваться следующие подходы:

* Обратная инженерия. Применяется с целью получения ПО на языке ассемблера или на языке высокого уровня.
* Анализ двоичного кода. Предполагает наличие анализирующего приложения, которое читает собранное ПО и просматривает его с применением некоторых эвристических правил.
* Тестирование нагрузкой, или стрессовое тестирование. Используется набор файлов сценариев, которые посылают ПО разнообразные входные данные различного размера и структуры.

1. **Программные измерительные мониторы**

Программные измерительные мониторы (ПИМ) - это совокупность команд или программ, выполняемых исключительно с целью проведения измерений параметров ресурсов.

При этом ПИМ собирает данные о ходе выполнения программ и накапливает их в памяти.

Встроенные ПИМ включаются в состав ОС на этапе генерации.

Недостатки ПИМ

1)нет программ автоматического анализа их протоколов;

2) низкая точность сопоставления ресурса и задания, которое его потребляет;

3) нет в результатах измерений желаемого уровня детализации (нет структуры потребления ресурсов).

Автономные ПИМ

Большинство современных мониторов специализированные и являются

профилировщиками (анализатор кода выполнения программ).

1. **Защита программ от исследования**

Основными инструментами для исследования программ являются

* дизассемблеры
* отладчики.

Дизассемблирование  - это получение из исполняемого кода программы код на языке ассемблера.

Дизассемблер - программа, осуществляющая дизассемблирование.

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.

Универсальным методом противодействия дизассемблированию программы является шифрование.

Методы защиты программ от дизассемблирования

1. Динамическое шифрование и многопроходная расшифровка кода.

2.Методы, использующие вместе с шифрованием архивирование программного кода.

3.Методы, основанные на динамическом изменении кода программы в процессе выполнения.

Для взлома механизмов защиты применяются мощные эмулирующие отладчики. Они самостоятельно (без помощи процессора) интерпретируют и выполняют команды исследуемой программы.

1. **Исследование кода вредоносных программ**

Вредоносный код – это компьютерный скрипт, специально разработанный для создания и эксплуатации уязвимостей в системе.

С его помощью можно воздействовать как на масштабные компоненты системы (например, сетевую инфраструктуру), так и на более мелкие, такие как приложения и программы для компьютеров и мобильных устройств.

Стандартный путь для проникновения вредоносного кода в систему – через зараженный сайт или вредоносную ссылку/вложение в электронном письме.

Угроза может исходить не только из очевидно вредоносного источника, но и из вполне легитимного

*Дополнительные меры безопасности*

* Установить ПО для защиты от скриптов,
* Быть внимательным при переходе по ссылкам и загрузке вложений.
* Активировать защиту от всплывающих окон в браузере.
* Не использовать вход с правами администратора для ежедневных задач.
* С осторожностью использовать любое общедоступное соединение для передачи данных.
* Установить и правильно настроить файервол

*Типы ВПО*

•программы-вымогатели,

•шпионское ПО

•ВПО для удаленного управления

•Загрузчики (downloader)

•Майнеры (miner)

•Банковские трояны (banking trojan) \

•Рекламное ПО (adware).

1. **Планирование ревьюирования.**

Планирование:

* Определение целей
* Выбор участников
* Составление графика
* Определение методов

Этапы планирования ревьюирования:

* Инициализация. Руководитель проекта или его заместитель запрашивает из базы, хранящей все данные проекта, список объектов, готовых к инспекции, выбирает объект инспекции, затем назначает участников формальной инспекции: автора, ведущего и одного или нескольких инспекторов.
* Проверка готовности документов. Ведущий проверяет, что инспектируемые документы размещены в базе данных проекта, а их статус соответствует готовности к формальной инспекции. Если это не так, инспекция откладывается.
* Изменение статуса документов. Ведущий изменяет статус инспектируемых документов так, чтобы отметить факт начала инспекции и ограничить доступ к инспектируемой документации.
* Заполнение бланка инспекции. Ведущий копирует из базы данных проекта бланк инспекции и заносит в него идентификаторы инспектируемых и исходных документов и номера их версий, список участников с указанием их ролей и дату фактического начала процесса инспекции.
* Оценка времени. Ведущий оценивает время, необходимое инспекторам для подготовки, и продолжительность обсуждения. Время, отводимое на этап подготовки, не может быть менее одного часа.
* Определение даты, времени и места обсуждения. Если оно будет проходить в форме собрания, ведущий определяет дату, время и место. При этом может потребоваться согласование с другими участниками инспекции.