[ВВЕДЕНИЕ 3](#_Toc1)

[1. Методы организации работы в команде разработчиков. Системы контроля версий 5](#_Toc2)

[2. Цели, задачи, этапы и объекты ревьюирования. Планирование ревьюирования 7](#_Toc3)

[3. Цели, корректность и направления анализа программных продуктов. Выбор критериев сравнения 9](#_Toc4)

[4. Представление результатов сравнения. Примеры сравнительного анализа программных продуктов 11](#_Toc5)

[5. Цели, задачи и методы исследования программного кода 13](#_Toc6)

[6. Измерительные методы оценки программ: назначение, условия применения. 15](#_Toc7)

[7. Корректность программ. Эталоны и методы проверки корректности 17](#_Toc8)

[8. Метрики, направления применения метрик. Метрики сложности. Метрики стилистики. 19](#_Toc9)

[9. Исследование программного кода на предмет ошибок и отклонения от алгоритма 21](#_Toc10)

[10. Программные измерительные мониторы 22](#_Toc11)

[11. Применение отладчиков и дизассемблера 24](#_Toc12)

[12. Зашита программ от исследования 26](#_Toc13)

[13. Исследование кода вредоносных программ 28](#_Toc14)

[ВЫВОД 30](#_Toc15)

ВВЕДЕНИЕ

1. В современном обществе программное обеспечение стало неотъемлемой частью повседневной жизни, влияя на все сферы деятельности — от автоматизации бизнес-процессов до обеспечения безопасности критически важных инфраструктур. В условиях постоянного усложнения программных решений и быстрого развития технологий особое значение приобретает качество разрабатываемого ПО. Одним из ключевых механизмов его обеспечения является ревьюирование программных модулей.
2. Целью учебной практики по профессиональному модулю "Ревьюирование программных модулей" стало освоение основ командной разработки, изучение систем контроля версий (в частности, Git и GitHub), а также знакомство с целями, задачами и этапами процесса ревью кода. В ходе практики была получена возможность глубже понять важность планирования ревью, выбора объективных критериев оценки, а также методов анализа программных продуктов на соответствие заданным требованиям.
3. Особое внимание в рамках практики уделялось причинно-следственным связям в анализе программ, изучению эталонов корректности, методам проверки работоспособности и применению измерительных подходов, таких как метрики сложности и стилистики кода. Были получены как теоретические знания, так и практические навыки, включая работу с отладчиками и дизассемблерами, в частности с инструментом Ghidra, для выявления ошибок и отклонений от заданного алгоритма.
4. Значимую роль сыграло исследование программного кода, в том числе вредоносного, что позволило лучше понять существующие угрозы в области информационной безопасности и методы защиты программ от анализа и обратной разработки. Применение программных

измерительных мониторов стало важным компонентом практики: они

1. позволили отслеживать поведение программ в реальном времени, анализировать производительность и выявлять потенциальные уязвимости.
2. В современной разработке программного обеспечения ревью кода играет ключевую роль, способствуя:
3. повышению качества и надёжности программного кода;
4. раннему выявлению и устранению ошибок;
5. распространению знаний внутри команды;
6. соблюдению единых стандартов написания кода.
7. Освоение навыков ревью является важным этапом профессионального становления разработчика, способствуя развитию технической экспертизы и навыков критического мышления.
8. Полученные знания и практический опыт будут способствовать успешному выполнению задач в дальнейшей профессиональной деятельности, связанной с разработкой, тестированием и обеспечением качества программных продуктов.
9. 1. Методы организации работы в команде разработчиков. Системы контроля версий
10. Для эффективной совместной разработки программного обеспечения были изучены современные подходы к управлению проектами и взаимодействию в команде, такие как Agile, Scrum и Kanban. Эти гибкие методологии позволяют оперативно адаптироваться к изменяющимся требованиям, повышать прозрачность процессов и улучшать коммуникацию между участниками проекта.
11. Особое внимание было уделено системам контроля версий — таким как Git и платформам на его основе, например

**Методы организации работы в команде разработчиков.**

Ознакомился с основными аспектами организации работы в команде разработчиков, а также изучил системы контроля версий, такие как Git и GitHub, которые играют важную роль в современных процессах разработки программного обеспечения

В рамках практики были рассмотрены различные подходы к управлению командной работой. Наиболее популярными методологиями стали Agile, Scrum и Kanban — гибкие подходы, позволяющие оперативно адаптироваться к изменяющимся требованиям и более эффективно управлять проектами

Можно выделить несколько ключевых моментов:

Роли в команде: В каждой команде разработки присутствуют определённые роли — разработчик, тестировщик, аналитик, Scrum-мастер и другие. Чёткое разделение обязанностей между участниками способствует устойчивой и слаженной работе всей команды

Коммуникация: Эффективное взаимодействие внутри коллектива напрямую влияет на производительность. Использование регулярных встреч, чатов и других коммуникационных инструментов позволяет заранее выявлять проблемы и согласовывать действия участников проекта

Планирование задач: Для успешного управления проектами необходимо применять методы планирования, включая оценку трудоёмкости задач и распределение ресурсов. Это является важным этапом в обеспечении своевременного и качественного выполнения проекта

**Системы контроля версий**

Кроме того, мы подробно изучили системы контроля версий (СКВ), которые являются важной частью повседневной работы разработчиков. Такие системы позволяют отслеживать все изменения в коде, управлять различными версиями проекта и организовывать эффективное взаимодействие в ходе совместной разработки. В рамках практики основное внимание было уделено одной из самых популярных и удобных систем — Git.

**Git** — это распределённая система контроля версий, предназначенная для работы нескольких разработчиков над одним проектом одновременно. Она обеспечивает сохранение полной истории изменений, а также возможность восстановления предыдущих версий кода при необходимости. В процессе изучения были рассмотрены базовые команды Git, принципы его работы, такие как создание и управление ветками, объединение изменений и разрешение конфликтов, возникающих при параллельной разработке.

**Инструменты:** Git/GitHub

1. Цели, задачи, этапы и объекты ревьюирования. Планирование ревьюирования

Изучены основные цели, задачи, этапы и объекты ревьюирования, а также процессы планирования, которые способствуют качественной оценке программного обеспечения. Ревьюирование является мощным инструментом для повышения надежности и безопасности разрабатываемых решений, что особенно важно в контексте современных требований к качеству ПО.

**Цели и задачи ревьюирования**

Основными целями ревьюирования являются:

* Выявление и устранение ошибок на ранних этапах разработки, что существенно снижает затраты на исправление в дальнейшем.
* Повышение качества и надежности программных продуктов.
* Обмен знаниями между членами команды и улучшение коммуникации.

Задачи ревьюирования включают:

* Анализ кода на предмет соответствия установленным стандартам.
* Оценка архитектуры и дизайна программного обеспечения.
* Проверка на соответствие требованиям и спецификациям.

Этапы и объекты ревьюирования

Ревьюирование проходит через несколько этапов:

1. Подготовка: выбор объектов обсуждения и формирование команды ревьюеров.

2. Проведение ревью: анализ кода, выявление проблем и формирование замечаний.

3. Обсуждение результатов: совместное обсуждение выявленных недостатков и предложений по их устранению.

4. Составление отчета по результатам ревью, который включает в себя выявленные проблемы и рекомендации.

Объектами ревьюирования могут быть:

* Исходный код программ.
* Документация.
* Архитектурные решения.

**Планирование ревьюирования**

Правильное планирование ревьюирования критически важно для достижения поставленных целей. В ходе практики изучил ключевые аспекты планирования процесса:

* Определение объема ревью: выбор компонентов или модулей, которые подлежат анализу.
* Установление сроков и поведения ревью: назначение времени для встреч и обсуждений, а также распределение ролей среди участников.
* Упрощение коммуникации между членами команды, использование специализированных инструментов для отслеживания замечаний и предложений.

1. Цели, корректность и направления анализа программных продуктов. Выбор критериев сравнения

Анализ программного обеспечения является важным этапом, позволяющим выявить его сильные и слабые стороны, оценить соответствие требованиям и повысить общее качество разработки.

**Цели анализа программных продуктов**

Основные цели анализа программных продуктов включают:

1. Оценка качества программного обеспечения: Определение уровня качества и надежности продуктов, что позволяет сделать выводы о их пригодности для использования.

2. Выявление ошибок и уязвимостей: Анализ помогает обнаружить потенциальные проблемы на ранних стадиях, что минимизирует риски и затраты на исправление в будущем.

3. Сравнительный анализ: Позволяет проводить сравнение различных решений и выбирать лучшее на основании сформулированных критериев.

**Корректность анализа программных продуктов**

Корректность анализа программных продуктов основывается на следующих аспектах:

* Актуальность тестируемых характеристик: Необходимо выбирать параметры, которые актуальны и значимы для данного типа программного обеспечения.
* Следование установленным стандартам и методикам: Рекомендуется использовать проверенные методики, такие как тестирование на зависимость, функциональное тестирование и статический анализ кода. Это позволит избежать субъективности в оценках.

**Направления анализа программных продуктов**

В практике мы выделили несколько направлений анализа программных продуктов:

* Функциональный анализ: Оценка соответствия функциональности продукта требованиям и спецификациям.
* Статический анализ: Изучение исходного кода на предмет надежности, читаемости и структурной корректности, без запуска программного обеспечения.
* Динамический анализ: Тестирование программы в реальном времени для оценки ее производительности и поведения в процессе эксплуатации.

**Выбор критериев сравнения**

В ходе практики были изучены ключевые аспекты организации работы в команде разработчиков, а также рассмотрены системы контроля версий, такие как Git и GitHub, которые являются важной частью современной разработки программного обеспечения.

Были проанализированы различные подходы к управлению проектами и взаимодействию внутри команд. Среди наиболее популярных методологий выделяются Agile, Scrum и Kanban — они позволяют гибко реагировать на изменения, улучшать коммуникацию и повышать общую эффективность работы.

Среди основных наблюдений можно выделить следующие:

Ролевое распределение: В каждой команде участники выполняют определённые функции: разработчики, тестировщики, аналитики, Scrum-мастеры и другие. Чёткое разделение ролей и зон ответственности способствует слаженной работе и высокой результативности.

Эффективная коммуникация: Успешное взаимодействие между членами команды играет ключевую роль в достижении целей проекта. Регулярные встречи, использование мессенджеров и других инструментов облегчают обмен информацией и оперативное решение возникающих вопросов.

Планирование и управление задачами: Для успешной реализации проектов необходимо использовать методы планирования, такие как оценка трудоёмкости задач и распределение ресурсов. Это позволяет заранее прогнозировать сроки выполнения работ и минимизировать риски срыва графика.

**Представление результатов сравнения**

Результаты сравнительного анализа должны быть представлены в ясной и структурированной форме. Основными способами представления являются:

* Табличный формат: Позволяет наглядно сравнить различные характеристики программных продуктов. В таблицах можно указать такие параметры, как функциональность, производительность, удобство использования, стоимость, степень поддержки и другие ключевые метрики.
* Графики и диаграммы: Визуализированная информация помогает лучше понять отличия между продуктами. Например, можно использовать линейные графики для иллюстрации производительности программ или столбчатые диаграммы для сравнения стоимости лицензий.
* Обзорные отчёты: Предоставляют детализированное описание каждого продукта, включая анализ достоинств и недостатков, рекомендации по использованию и области применения.

**Примеры сравнительного анализа программных продуктов**

В ходе практики были проведены несколько примеров сравнительного анализа программных продуктов, среди которых:

1. Сравнительный анализ систем контроля версий (Git vs SVN):

* Таблица сравнения:

|  |  |  |
| --- | --- | --- |
| Характеристика | Git | SVN |
| Архитектура | Распределенная | Клиенто-серверная |
| Поддержка веток | Да | Ограниченная |
| Производительность | Высокая | Средняя |
| Простота использования | Умеренно сложная | Простая |

* График производительности: Отображение времени выполнения операций(коммит, слияние) для каждой системы.

2. Сравнительный анализ браузеровпо соответствующим характеристикам

3. Сравнительный анализ офисных пакетов по соответствующим характеристикам

3. Сравнительный анализ средств просмотра видео по соответствующим характеристикам

1. Цели, задачи и методы исследования программного кода

Ознакомился с основными целями и задачами, а также с различными методами исследования программного кода. Правильное исследование кода — ключевой аспект обеспечения качества программного обеспечения, который позволяет выявить и устранить потенциальные проблемы ещё на ранних этапах разработки.

**Цели исследования программного кода**

Основные цели исследования программного кода включают:

* Выявление и устранение дефектов: Раннее обнаружение ошибок и уязвимостей в коде позволяет свести к минимуму затраты на исправление в будущем.
* Повышение качества кода: Целью является не только поиск ошибок, но и оптимизация структуры, стилевых решений и производительности программного обеспечения.
* Соблюдение стандартов разработки: Исследование кода помогает убедиться в соответствии кода принятым стандартам и практикам, что способствует улучшению его читаемости и поддерживаемости.

**Задачи исследования программного кода**

Задачи, которые ставятся перед исследованием программного кода, могут включать:

* Анализ соответствия кода требованиям спецификации и функциональным требованиям.
* Оценка структуры и архитектуры программного обеспечения.
* Выявление "технического долга" и анализ возможностей его сокращения.
* Проверка безопасности кода, идентификация потенциальных уязвимостей.

**Методы исследования программного кода**

В ходе практики были изучены различные методы анализа программного кода. Среди них можно выделить следующие:

Статический анализ — представляет собой метод исследования кода без его выполнения. С помощью специализированных инструментов выявляются потенциальные ошибки, нарушения стандартов кодирования и возможные уязвимости безопасности.

Динамический анализ — заключается в исследовании поведения программы во время выполнения. Данный подход позволяет обнаруживать проблемы с производительностью, логические ошибки и сбои, которые проявляются только при реальном запуске приложения.

Код-ревью — это процесс коллективного рассмотрения написанного кода. Его участники проверяют реализацию на соответствие требованиям, читаемость и эффективность решений. Такая практика способствует выявлению скрытых дефектов и развитию профессиональных навыков в команде.

Unit-тестирование — предполагает проверку отдельных модулей или функций программы с целью обеспечения их корректной работы. Написание тестовых сценариев помогает подтвердить, что код ведёт себя ожидаемым образом при различных входных данных.

5. Измерительные методы оценки программ: назначение и условия применения

Было проведено исследование измерительных методов оценки программного обеспечения. Эти методы играют важную роль в обеспечении высокого уровня качества и надёжности разрабатываемых программных продуктов. Они позволяют объективно оценивать такие характеристики, как читаемость, сложность, производительность и поддерживаемость кода, а также выявлять направления для дальнейшего улучшения.

**Назначение измерительных методов оценки программ**

Измерительные методы предназначены для:

* Оценки качества программного обеспечения: Позволяют выявить уровень соответствия программных продуктов установленным стандартам и требованиям, что способствует повышению доверия пользователей к продуктам.
* Выявления проблемных областей: Помогают определить участки кода или функциональности, требующие улучшений или исправлений.
* Сравнительного анализа: Обеспечивают возможность сравнивать различные версии одного и того же продукта или различные продукты на основе объективных метрик, что упрощает процесс выбора оптимального решения.

**Условия применения измерительных методов оценки программ**

Применение измерительных методов возможно при соблюдении следующих условий:

* Наличие четких критериев качества: Для успешной оценки необходимо определение конкретных критериев, по которым будет проводиться измерение(например, производительность, стабильность, безопасность).
* Доступность необходимых инструментов и методик: Нужно использовать надежные инструменты для проведения измерений, такие как статические анализаторы, профилировщики производительности и системы для автоматизированного тестирования.
* Условия репрезентативности: Среда тестирования должна быть как можно ближе к реальным условиям эксплуатации программы. Это поможет получить более точные результаты оценок и выявить потенциальные проблемы.
* Определение временных рамок: Необходимо установить временные границы для проведения измерительных процессов, чтобы получить актуальные данные, которые будут отражать текущее состояние программного обеспечения.

1. Корректность программ. Эталоны и методы проверки корректности

Ознакомился с концепцией корректности программ. Корректность программного обеспечения является критически важной характеристикой, так как она определяет, насколько программа отвечает заданным спецификациям и требованиям. Исследование методов проверки корректности и применение эталонов позволяет повысить уровень доверия к программным продуктам.

**Корректность программ**

Корректность программы можно определить как её соответствие заранее установленным требованиям и ожиданиям, включая как функциональные, так и нефункциональные характеристики. Обеспечение корректности программ требует системного подхода и применения различных методов исследования.

**Эталоны корректности**

Эталоны корректности характеризуются набором требований и спецификаций, которым программа должна соответствовать. Основные типы эталонов включают:

* Технические спецификации: Документация, в которой описываются функциональные и нефункциональные требования к системе.
* Математические модели: Формальные модели, которые используются для определения и формализации требований к корректности.
* Стандарты разработки: Определенные индустриальные или организационные практики и правила, которым следуют при разработке программного обеспечения.

**Методы проверки корректности**

В рамках практики были изучены несколько методов проверки корректности программ:

* Статический анализ: Этот метод заключается в анализе исходного кода без его выполнения с целью выявления потенциальных проблем, нарушений стиля и несоответствий требованиям спецификации. Инструменты статического анализа помогают находить ошибки на этапе разработки, что способствует экономии времени и ресурсов.
* Динамический анализ: В отличие от статического, этот метод включает выполнение программы для проверки её поведения. Динамический анализ позволяет выявлять ошибки, которые могут проявляться только в процессе выполнения, включая проблемы производительности и утечки памяти.
* Формальная верификация: Этот метод применяется для математического доказательства корректности программ. Формальные методы обеспечивают высокий уровень уверенности в корректности программ, особенно для критически важных систем.
* Модульное тестирование: Тестирование отдельных модулей программы на предмет их соответствия спецификациям. Это помогает убедиться, что каждая часть программы работает правильно и в соответствии с заданными требованиями.

1. Метрики, направления применения метрик. Метрики сложности. Метрики стилистики.

В ходе работы было изучено значение метрик в оценке качества программного обеспечения. Использование метрик позволяет количественно анализировать различные характеристики кода, выявлять потенциальные проблемы и повышать его общее качество. В данном отчёте рассматриваются основные направления применения метрик, а также их классификация, включающая метрики сложности кода и метрики стиля программирования.

**Направления применения метрик**

Метрики применяются в нескольких направлениях:

* Оценка качества кода: Позволяют выявлять участки кода, требующие улучшений, и обеспечивают объективную оценку его качества.
* Управление техническим долгом: Помогают определить и оценить технический долг, что позволяет планировать мероприятия по его снижению.
* Сравнительный анализ: Оценивают эффективность различных подходов к разработке или сравнивают разные версии программного обеспечения.
* Прогнозирование затрат: Включают метрики, позволяющие оценить временные и ресурсные затраты на разработку и поддержку программного продукта.

**Метрики сложности**

Метрики сложности используются для оценки сложности кода, что напрямую влияет на его читабельность и сопровождаемость. К основным метрикам сложности относятся:

* Метрика цикломатической сложности: Измеряет число линейно независимых путей через программу. Чем выше это значение, тем сложнее код для тестирования и сопровождения.
* Количество строк кода(LOC): Простой, но полезный показатель, показывающий объем кода. Однако высокая метрика LOC не всегда отражает сложность, так как может зависеть от стиля программирования.
* Число коммитов и изменений: Это метрики, которые позволяет оценить, сколько раз код изменялся, что может быть показателем сложности проекта.

**Метрики стилистики**

Метрики стилистики фокусируются на оформлении и структуре кода, что также влияет на его понимание и поддержку. К ним относятся:

* Индексы согласованности именования: Измеряют, насколько последовательно используются имена переменных и функций, что важно для читаемости кода.
* Коэффициенты дублирования кода: Оценка количества повторяющихся фрагментов кода. Высокий уровень дублирования может свидетельствовать о низком качестве кода и необходимости его рефакторинга.
* Стандарты оформления кода: Проверка на соответствие установленным стандартам оформления, что способствует унификации кода и облегчает его поддержку командой разработчиков.

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

Изучил методы и подходы, которые позволяют исследовать программный код на предмет ошибок и отклонений от заданного алгоритма. Это исследование является критически важным для обеспечения качества программного обеспечения и достижения его соответствия установленным требованиям.

**Методы исследования кода**

Исследование программного кода включает различные методики и инструменты, которые помогают выявить ошибки и несоответствия:

* Статический анализ кода: Этот метод включает проверку кода без его выполнения, что позволяет выявить потенциальные проблемы, такие как синтаксические ошибки, несоответствия правилам стиля и нарушения логики программы.
* Динамический анализ: В отличие от статического, этот метод включает выполнение программы для наблюдения за её поведением. Динамическое тестирование позволяет находить ошибки, которые могут проявляться только в процессе выполнения, а также выявлять проблемы с производительностью.
* Тестирование на основе спецификаций: Этот подход включает проверку соответствия результатов работы программы заданным спецификациям и алгоритмам. С помощью модульного и интеграционного тестирования можно удостовериться, что каждая часть кода работает корректно и результаты соответствуют ожиданиям.
* Анализ отклонений от алгоритма: Этот метод направлен на выявление случаев, когда программа отклоняется от заданного алгоритма. Студенты изучают, как логика кода может привести к ошибкам, и используют отладку, чтобы проанализировать причины несоответствий.

1. Программные измерительные мониторы

Изучил концепцию программных измерительных мониторов. Эти инструменты играют важную роль в процессе разработки программного обеспечения, поскольку позволяют отслеживать производительность приложения, выявлять узкие места и обеспечивать качество кода. В данном отчете рассматриваются основные аспекты работы программных измерительных мониторов и их применение в практике ревьюирования кода.

**Определение программных измерительных мониторов**

Программные измерительные мониторы(или метрики производительности) — это программные инструменты, которые собирают и анализируют данные о функционировании программного обеспечения. Они помогают разработчикам получить представление о параметрах производительности, таких как скорость выполнения, использование ресурсов и частота сбоев. Использование таких мониторов позволяет выявить проблемы и оптимизировать код еще на этапе разработки.

**Типы метрик**

* Метрики производительности: Они измеряют время выполнения программных операций и скорость обработки данных. Метрики могут включать время отклика системы, время загрузки страниц и задержки при выполнении запросов.
* Метрики использования ресурсов: Эти метрики отслеживают использование процессора, памяти, дискового пространства и сети. Это позволяет выявить участки кода, которые могут вызывать избыточное использование ресурсов.
* Метрики стабильности: Они определяют частоту сбоев и ошибок в работе программы. Высокая частота ошибок сигнализирует о необходимости доработки кода и улучшения его качества.

**Подходы к применению**

Для эффективного применения программных измерительных мониторов были рассмотрены различные подходы:

Непрерывный мониторинг — заключается во встраивании средств отслеживания в процессы разработки и тестирования, что обеспечивает сбор данных в режиме реального времени. Такой подход позволяет оперативно выявлять отклонения и отслеживать показатели на всех этапах жизненного цикла продукта

Анализ данных — предполагает использование специализированных инструментов для обработки, интерпретации и визуализации накопленной информации. Это помогает выявлять тенденции, оценивать эффективность решений и принимать обоснованные управленческие решения

Регулярные ревью — включают систематический анализ как самого кода, так и его производительности. Постоянная проверка метрик и характеристик программного обеспечения способствует поддержанию высокого уровня качества и оперативному реагированию на потенциальные проблемы

1. Применение отладчиков и дизассемблера

Изучил важность и применение отладчиков и дизассемблеров для анализа программного кода. Эти инструменты позволяют разработчикам глубже понять работу программ, выявлять и исправлять ошибки, а также проводить анализ производительности. В данном отчете рассматриваются основные аспекты работы с отладчиками и дизассемблерами, в частности на примере инструмента Ghidra.

**Определение отладчиков и дизассемблеров**

Отладчики представляют собой инструменты, позволяющие разработчикам выполнять программу пошагово, наблюдать за состоянием переменных и отслеживать последовательность выполнения кода. Дизассемблеры, такие как Ghidra, выполняют обратное преобразование — переводят машинный код в понятный человеку ассемблерный код, что упрощает анализ и изучение работы программы на низком уровне.

**Применение отладчиков**

* Выявление ошибок: Отладчики позволяют выявлять ошибки, наблюдая за состоянием приложения в реальном времени. Это важно для поиска логических ошибок и проблем с производительностью.
* Анализ потока выполнения: С помощью отладчика студенты могут отслеживать, какие функции и методы вызываются, и анализировать, как изменяются переменные во время выполнения программы.
* Тестирование и профилирование: Отладчики помогают в тестировании отдельных модулей и функций, а также в профилировании производительности, выявляя узкие места в коде.

**Применение дизассемблера Ghidra**

* Анализ бинарников: Ghidra позволяет проводить статический анализ бинарных файлов, что полезно для изучения работы защищенных или закрытых программ, когда исходный код недоступен.
* Исследование уязвимостей: Дизассемблирование позволяет находить уязвимости в коде, анализируя его на более низком уровне, что способствует повышению безопасности программ.
* Реверс-инжиниринг: Ghidra предоставляет возможности для реверс-инжиниринга, что может быть полезным для изучения старых или устаревших систем, а также для создания патчей и улучшений к программному обеспечению.

1. Зашита программ от исследования

Ознакомился с важными аспектами защиты программного обеспечения от анализа и исследования. Современные программы, особенно те, которые содержат интеллектуальную собственность или чувствительные данные, должны быть защищены от реверс-инжиниринга и других видов анализа. Данная часть отчёта посвящена методам и инструментам, использующимся для повышения безопасности программ и защиты их от несанкционированного исследования.

**Определение защиты программ**

Защита программ от исследования включает в себя различные техники и подходы, направленные на затруднение анализа кода и предотвращение его несанкционированного использования. Это может быть реализовано через программные защитные механизмы, криптографические методы и обфускацию кода.

**Методы защиты**

* Обфускация кода — представляет собой метод преобразования исходного или машинного кода с целью затруднить его анализ и понимание. Такой подход широко используется для защиты алгоритмов, бизнес-логики и предотвращения реверс-инжиниринга с помощью специализированных инструментов — обфускаторов.
* Шифрование — заключается в использовании криптографических алгоритмов для защиты передаваемых и хранимых данных. Зашифрованная информация становится недоступной без соответствующего ключа, что особенно актуально при обработке конфиденциальной пользовательской информации.
* Защита от отладчиков — подразумевает внедрение механизмов, которые либо блокируют работу отладочных инструментов, либо усложняют их эффективное использование. Например, программа может проверять своё окружение на наличие отладчика и при его обнаружении изменять логику работы или принудительно завершать выполнение.
* Лицензионные механизмы: Использование различных систем лицензирования способствует контролю доступа к программному обеспечению и гарантирует, что только авторизованные пользователи могут использовать продукт.

1. Исследование кода вредоносных программ

Получил знания о методах анализа и исследования вредоносного программного обеспечения(Malware), а также о мерах безопасности при анализе ВПО. Понимание работы вредоносных программ и их структуры позволяет разработчикам и специалистам по безопасности выявлять уязвимости и защищать информационные системы. В данном отчете рассматриваются основные аспекты исследования кода вредоносных программ, их классификация и методы анализа.

**Определение вредоносных программ**

Вредоносные программы — это разновидность программного обеспечения, созданная с целью причинения вреда компьютерам, пользователям или сетевым системам. Они могут существовать в различных формах, таких как вирусы, черви, трояны, шпионские и рекламные приложения. Анализ их кода имеет важное значение для изучения механизмов их действия и создания надёжных методов противодействия им.

Если нужно — могу адаптировать под научную статью, презентацию или техническую документацию.

**Методы исследования кода вредоносных программ**

* Статический анализ: Этот метод включает изучение бинарного и исходного кода без его выполнения. Студенты изучают структуры кода, включая функции, вызовы API и зависимости, для понимания логики работы вредоносной программы и определения ее поведения.
* Динамический анализ: В этом методе исследуется поведение программы во время ее выполнения. Используют отладчики и эмуляторы, чтобы отслеживать вызовы функций, изменения в файловой системе и сетевую активность. Динамический анализ позволяет получить более полное представление о действиях вредоносной программы в реальных условиях.
* Использование дизассемблеров: Инструменты, такие как Ghidra или IDA Pro, помогают преобразовывать машинный код обратно в ассемблерный, что позволяет исследователям понимать структуру и логику работы программы на низком уровне.
* Поиск сигнатур: Этот метод основан на идентификации уникальных паттернов или сигнатур, характерных для определенного вредоносного ПО. Базы данных антивирусных программ содержат такие сигнатуры, что упрощает обнаружение уже известных угроз.

ВЫВОД

В ходе учебной практики по профессиональному модулю «Ревьюирование программных модулей» были получены и закреплены важные теоретические знания и практические навыки в области анализа, оценки и улучшения качества программного обеспечения. Было изучено множество аспектов, начиная с организации работы в команде разработчиков и использования систем контроля версий (в частности Git/GitHub), и заканчивая исследованием программного кода, включая анализ вредоносного ПО и защиту программ от несанкционированного исследования.

Особое внимание было уделено процессам ревью кода как одному из ключевых инструментов обеспечения надежности, безопасности и соответствия стандартам программных продуктов. Изучены цели, задачи, этапы и методы ревьюирования, корректность анализа, выбор критериев сравнения, а также применение метрик сложности и стилистики для объективной оценки качества кода. Также был освоен практический подход к использованию отладчиков и дизассемблеров (например, Ghidra) для выявления ошибок и анализа поведения программ на низком уровне.

Кроме того, были рассмотрены современные измерительные методы и программные мониторы, позволяющие отслеживать производительность, находить узкие места и потенциальные уязвимости. Особый интерес вызвало исследование вредоносных программ и изучение методов их анализа, что расширило понимание угроз в сфере информационной безопасности.

Практика позволила не только углубить техническую экспертизу, но и развить навыки критического мышления, необходимые при работе с программными продуктами. Полученные знания и опыт будут способствовать успешному выполнению профессиональных задач в будущем, направленных на создание качественного, безопасного и надежного программного обеспечения.