СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 2](#_Toc200139134)

[**1.** Методы организации работы в команде разработчиков. Системы контроля версий 4](#_Toc200139135)

[**2.** Цели, задачи, этапы и объекты ревьюирования. Планирование ревьюирования 6](#_Toc200139136)

[**3.** Цели, корректность и направления анализа программных продуктов. Выбор критериев сравнения 8](#_Toc200139137)

[**4.** Представление результатов сравнения. Примеры сравнительного анализа программных продуктов 10](#_Toc200139138)

[**5.** Цели, задачи и методы исследования программного кода 12](#_Toc200139139)

[**6.** Измерительные методы оценки программ: назначение, условия применения. 14](#_Toc200139140)

[**7.** Корректность программ. Эталоны и методы проверки корректности 16](#_Toc200139141)

[**8.** Метрики, направления применения метрик. Метрики сложности. Метрики стилистики. 18](#_Toc200139142)

[**9.** Исследование программного кода на предмет ошибок и отклонения от алгоритма 20](#_Toc200139143)

[**10.** Программные измерительные мониторы 21](#_Toc200139144)

[**11.** Применение отладчиков и дизассемблера 23](#_Toc200139145)

[**12.** Зашита программ от исследования 25](#_Toc200139146)

[**13.** Исследование кода вредоносных программ 27](#_Toc200139147)

[ВЫВОД 29](#_Toc200139148)

ВВЕДЕНИЕ

В современной разработке программного обеспечения владение инструментами контроля версий и навыками эффективного программирования является неотъемлемой частью профессиональной деятельности. Данная практическая работа была посвящена освоению и применению двух ключевых технологий:

1. **Система контроля версий Git и платформа GitHub:** как фундаментальные инструменты для управления исходным кодом, организации командной работы, отслеживания изменений и обеспечения целостности проекта.
2. **Язык программирования Python:** как универсальный и мощный инструмент для решения практических задач, автоматизации процессов и реализации функциональности.

**Целью практики** являлось закрепление теоретических знаний на практике через выполнение конкретных задач, включающих:

* Создание и управление репозиторием на GitHub.
* Освоение базовых и продвинутых команд Git (clone, commit, push, pull, branch, merge, работа с .gitignore).
* Организацию совместной разработки с использованием механизмов ветвления (branching), пул-реквестов (Pull Requests) и разрешения конфликтов (merge conflicts).
* Разработку, отладку и рефакторинг кода на Python в рамках поставленных задач.
* Интеграцию процессов работы с кодом Python в Git-воркфлоу (коммиты изменений кода, работа с историей).
* Использование возможностей GitHub для управления проектом (Issues, Projects, Wiki, Actions для CI/CD).

**Актуальность** данной работы обусловлена тем, что комбинация Git/GitHub и Python составляет основу workflow огромного количества проектов — от небольших скриптов до крупных распределенных систем в областях Data Science, веб-разработки, DevOps и автоматизации. Умение грамотно применять эти инструменты в связке критически важно для успешной карьеры в IT.

В ходе практики были выполнены задачи, направленные на моделирование реального процесса разработки: от постановки задачи через Issue до реализации функциональности на Python, оформления изменений в виде коммитов и веток, проведения код-ревью через Pull Request и окончательного слияния кода в основную ветку.

**Основными задачами практики** стали:

* Настройка локального и удаленного (GitHub) рабочего окружения.
* Реализация конкретных алгоритмов или функциональности на Python согласно требованиям.
* Организация кода в соответствии с лучшими практиками (структура репозитория, PEP-8).
* Обеспечение эффективной командной работы (или имитации таковой) с использованием возможностей GitHub.
* Освоение практик Continuous Integration/Continuous Deployment (CI/CD) на базе GitHub Actions (при наличии соответствующего задания).

Данное введение определяет контекст, цели и ключевые направления выполненной практической работы, которая позволила получить и закрепить практические навыки, необходимые для дальнейшей учебной и профессиональной деятельности в области разработки ПО.

**Ключевые моменты этого введения:**

1. **Контекст:** Подчеркивает важность Git/GitHub и Python в современной разработке.
2. **Цель:** Четко формулирует, чего хотели достичь в рамках практики.
3. **Актуальность:** Объясняет, почему именно эта комбинация инструментов так важна.
4. **Связка инструментов:** Явно указывает, что работа велась одновременно с обоими инструментами и их интеграцией.
5. **Основные направления деятельности:** Перечисляет ключевые действия (работа с Git, программирование на Python, командная работа на GitHub, управление проектом).
6. **Конкретные задачи:** Детализирует, что именно делалось (настройка, написание кода, организация, CI/CD).
7. **Значимость:** Указывает на практическую ценность полученных навыков.
8. **Формальный стиль:** Подходит для учебного или отчетного документа.
9. Методы организации работы в команде разработчиков. Системы контроля версий

**Методы организации работы в команде разработчиков.**

Ознакомился с ключевыми аспектами организации работы в команде разработчиков, а также изучить системы контроля версий(в частности Git/GitHub), играющие важную роль в современном процессе разработки программного обеспечения.

В ходе практики мы изучили различные методы организации работы в команде, среди которых наиболее распространенными являются Agile, Scrum и Kanban. Эти методологии позволяют командам более гибко реагировать на изменения требований и эффективно управлять проектами.

Хочу выделить следующие ключевые моменты:

* Роли в команде: Каждую команду разработки составляют разные роли, такие как разработчик, тестировщик, аналитик и Scrum-мастер. Каждая роль имеет свои задачи и зоны ответственности, что помогает добиться оптимальной организационной структуры.
* Коммуникация: Эффективная коммуникация внутри команды способствует повышению продуктивности. Инструменты, такие как регулярные совещания и средства для обмена сообщениями, помогают поддерживать связь и раскрывать потенциальные проблемы на ранних стадиях.

3. Планирование и оценка задач: Использование методов планирования, таких как оценка трудоемкости задач и распределение ресурсов, есть обязательным шагом к эффективному управлению проектами.

**Системы контроля версий**

Также мы подробно ознакомились с системами контроля версий(СКВ), которые являются неотъемлемой частью работы разработчиков. СКВ позволяют отслеживать изменения в коде, управлять версиями и обеспечивать совместную работу над проектом. В практике мы рассматривали наиболее популярную и удобную систему: Git.

Git: Это распределенная система контроля версий, позволяющая разработчикам работать над одной и той же кодовой базой одновременно, при этом сохраняя возможность отката изменений и ведения истории проекта.

Были изучены основные команды этих систем, принципы их работы, таких как создание веток, слияние изменений и разрешение конфликтов.

**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**

При сравнении программных продуктов важно основывать свои выводы на объективных и измеримых критериях. К ним относятся:

- Метрики качества: Например, метрики сложности кода, метрики стиля.

- Производительность: Время отклика, загрузка системы и другие параметры, влияющие на пользовательский опыт.

- Надежность и тестируемость: Устойчивость к ошибкам, легкость в проведении тестов и исправление недостатков.

1. Представление результатов сравнения. Примеры сравнительного анализа программных продуктов

Изучил методы представления результатов сравнительного анализа программных продуктов. Сравнительный анализ позволяет не только выявить сильные и слабые стороны различных решений, но и помогает сделать обоснованный выбор в пользу конкретного продукта, основываясь на фактических данных и критериям оценки.

**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 рамках практики изучили различные методы исследования программного кода, среди которых:

* Статический анализ: Метод, который позволяет проверять исходный код без его выполнения. Используются специальные инструменты для выявления ошибок, несоответствий стандартам и уязвимостей.
* Динамический анализ: В отличие от статического, этот метод включает выполнение программы и оценку её поведения в реальных условиях. Динамический анализ помогает выявить проблемы производительности и ошибки, которые могут проявляться только при выполнении кода.
* Код-ревью: Процесс, в котором коллеги по команде анализируют код друг друга с целью выявления проблем и предложений по улучшению. Это не только помогает находить ошибки, но и способствует обмену знаниями в команде.
* Unit-тестирование: Метод, направленный на тестирование отдельных модулей или функций кода для проверки их корректности. Тесты помогают убедиться, что код работает как ожидается при различных условиях.

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

Исследовал измерительные методы оценки программного обеспечения. Эти методы играют важную роль в обеспечении качества и надежности программных продуктов, позволяя объективно оценивать их характеристики и выявлять области для 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.

Ознакомился с важностью метрик в оценке качества программного обеспечения. Метрики позволяют количественно оценивать различные аспекты кода, что способствует выявлению проблем и улучшению его характеристик. В данном отчете рассматриваются направления применения метрик, а также различные категории метрик, включая метрики сложности и метрики стилистики.

**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 использования программных измерительных мониторов изучили различные подходы:

* Непрерывный мониторинг: Этот метод включает интеграцию мониторов в процессы разработки и тестирования, позволяя собирать данные в реальном времени.
* Анализ данных: Применение специализированных аналитических инструментов для обработки и визуализации собранных данных. Это помогает разработчикам выявить закономерности и принять обоснованные решения по оптимизации.
* Регулярные ревью: Вместе с анализом данных, регулярные проверки кода и его производительности позволяют команде разработчиков поддерживать высокие стандарты качества и эффективно реагировать на возникшие 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 — это инструменты, которые позволяют разработчикам выполнять код построчно, просматривать значения переменных и отслеживать поток выполнения программы. Дизассемблеры, такие как Ghidra, преобразуют машинный код обратно в ассемблерный, что облегчает анализ и понимание низкоуровневой работы программы.

**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: Этот метод включает изменение исходного кода или скомпилированного бинарного файла таким образом, чтобы сделать его трудночитаемым и сложным для понимания. Применение обфускаторов помогает защитить бизнес-логики и алгоритмы от анализа и реверс-инжиниринга.
* Шифрование: Для защиты данных, передаваемых и хранящихся программой, широко применяются криптографические алгоритмы, которые шифруют информацию, делая ее непонятной без соответствующего ключа. Это особенно важно для защиты конфиденциальных данных пользователей.
* Защита от отладчиков: Внедрение механизмов, предотвращающих работу отладчиков, либо затрудняющих их использование, является еще одним способом защиты. Например, программы могут проверять наличие отладчиков и завершать работу или изменять поведение в их присутствии.
* 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 программы, которые имеют тенденцию наносить вред компьютерам, пользователям или сетям. Они могут иметь различные формы, включая вирусы, черви, трояны, шпионские и рекламные программы. Исследование кода этих программ необходимо для понимания механики их работы и разработки методов защиты от них.

**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.

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | Формула | Описание | Применение |
| Accuracy | (TP + TN) / (TP+TN+FP+FN) | Общая точность предсказаний | Базовый скрининг моделей |
| Precision | TP / (TP + FP) | Точность положительных прогнозов | Когда важны ложные срабатывания (FP) |

ВЫВОД

В ходе практики были освоены ключевые аспекты совместной работы с GitHub и разработки на Python:

1. Интеграция Git/GitHub с Python-проектами

- Организация структуры репозитория для Python-проектов (пакеты, модули, требования)

- Управление зависимостями

- Автоматизация тестирования и сборки с помощью GitHub Actions

2. Работа с кодом на Python

- Совместная разработка и рефакторинг Python-модулей

- Решение задач по обработке данных/автоматизации с использованием стандартных библиотек Python

- Соблюдение PEP-8 в коммитах и проведение код-ревью Python-скриптов

3. Командные практики

- Параллельная разработка фич в разных ветках с последующим слиянием через PR

- Контроль качества: юнит-тесты для Python-кода, проверка покрытия

- Документирование кода (docstrings, README.md с примерами использования)

Итог: Практика продемонстрировала контроля версий и Python-разработки. Освоенный стек позволяет эффективно реализовывать полноценные проекты — от локальной разработки до промышленного развертывания, с соблюдением стандартов командной работы и автоматизации жизненного цикла ПО.