|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации информационных технологий (МОСИТ)

|  |
| --- |
| **ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5** |
| **по дисциплине** |
| **«Тестирование и верификация программного обеспечения»** |

|  |  |
| --- | --- |
| Выполнили студенты группы ИКБО-04-22 | Егоров Л.А. |
| Принял ассистент | Петрова А.А. |

|  |  |  |
| --- | --- | --- |
| Практическая работа выполнена | «\_\_»\_\_\_\_\_\_\_202\_\_г. | *(подпись студента)* |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_202\_\_ г. | *(подпись руководителя)* |

Москва 2024

# 1 Статический анализ

Статический анализ кода — это процесс анализа исходного кода программы без его исполнения, направленный на поиск ошибок, уязвимостей, нарушений стандартов и других потенциальных проблем. Он проводится с использованием специальных инструментов (статических анализаторов), которые могут анализировать код на разных уровнях: от синтаксического до логического.

## 1.1 Анализ кода на языке Python

Для анализа был использован код электромагнитного алгоритма, который представлен в Листинге А.1.

Для анализа кода на языке Python использовалось три анализатора – MyPy, Pylint, flake8.

**MyPy** — это статический анализатор кода для Python, который проверяет типы данных в программах, написанных на Python. Он позволяет разработчикам использовать аннотации типов, чтобы явно указывать ожидаемые типы переменных, аргументов функций и возвращаемых значений. MyPy анализирует код и выявляет потенциальные ошибки, связанные с несоответствием типов, что помогает улучшить качество кода и предотвратить ошибки на этапе разработки.. Результат анализа этой программой представлен на Рисунке 1.1.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.1.1 – Анализ программой MyPy

**Pylint** — это статический анализатор кода для Python, который проверяет качество и стиль кода на соответствие стандартам PEP 8 и другим рекомендациям по написанию чистого и поддерживаемого кода. Pylint анализирует исходный код и генерирует отчеты о найденных ошибках, потенциальных проблемах, нарушениях стиля и других аспектах качества кода. Результат анализа этой программой представлен на Рисунке 1.1.2.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.1.2 – Анализ программой pylint

**Flake8** — это инструмент статического анализа кода для Python, который объединяет в себе несколько других инструментов для проверки качества и стиля кода. Flake8 включает в себя проверки на соответствие стандартам PEP 8, поиск синтаксических ошибок и потенциальных проблем в коде. Он объединяет функциональность трех основных компонентов: PyFlakes, pycodestyle (ранее известный как pep8) и mccabe. Результат анализа этой программой представлен на Рисунке 1.1.3.



Рисунок 1.1.3 – Анализ программой flake8

### 1.1.1 Добавление ошибок в код

В код исходной программы добавлено 5 ошибок:

1. Из функций были убраны строки документации (Рисунок 1.1.4).
2. Одна строка была сделана большой длины (больше 79 символов) (Рисунок 1.1.5).
3. Добавлена неиспользуемая переменная (Рисунок 1.1.6).
4. Допущена ошибка в типизации нескольких функций (Рисунок 1.1.7).
5. Добавлены лишние отступы между функциями (Рисунок 1.1.8).

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.1.4 – Отсутствие документации в функции

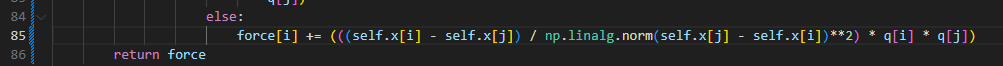


Рисунок 1.1.5 – Строка длиной больше 79 символов

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 1.1.6 – Добавление неиспользуемой переменной

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.1.7 – Ошибка в типизации результата функции

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 1.1.8 – Добавление лишних отступов

Результаты анализа представлены на Рисунках 1.1.9 - 1.1.11.

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Рисунок 1.1.1.1 – Анализ программой mypy

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 1.1.1.2 – Анализ программой pylint

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.1.1.3 – Анализ программой flake8

В результате программа mypy обнаружила ошибку, связанную с типизацией. Pylint больше указала на стилистические ошибки, а flake8 указала все ошибки, которые допущены в оформлении по PEP8.

## 1.2 Анализ кода на языке C

Для анализа взята программа на языке C, выполняющая операцию добавления строки в матрицу. Её код представлен в Листинге А.2.

Для анализа кода используются статические анализаторы cppcheck, clang-tidy и gcc analyzer.

**Cppcheck** — это легковесный статический анализатор кода, который выполняет анализ исходного кода на предмет ошибок, связанных с безопасностью, производительностью и другими аспектами качества кода. Он поддерживает языки C и C++ и предназначен для выявления потенциальных проблем, которые могут быть пропущены компилятором или другими инструментами. Результат анализа представлен на Рисунке 1.2.1.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.2.1 – Результат анализа cppcheck

**Clang-Tidy** — это инструмент статического анализа кода, встроенный в компилятор Clang, который выполняет анализ исходного кода на предмет ошибок, нарушений стиля, потенциальных уязвимостей и других аспектов качества кода. Clang-Tidy поддерживает языки C, C++ и Objective-C и предназначен для автоматизации процесса улучшения и очистки кода. Результат анализа представлен на Рисунке 1.2.2.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 1.2.2 – Результат анализа clang-tidy

**GCC -fanalyzer** — это опция компилятора GCC, которая включает встроенный статический анализатор кода. Этот анализатор выполняет глубокий анализ исходного кода на предмет ошибок, потенциальных уязвимостей и других аспектов качества кода. Он работает на этапе компиляции и может выявлять проблемы, которые могут быть пропущены другими инструментами. Результат анализа представлен на Рисунке 1.2.3.



Рисунок 1.2.3 – Результат анализа gcc -fanalyzer

# 2 Динамический анализ

## 2.1 Анализ кода на языке Python

Для динамического анализа кода на языке Python использованы два анализатора кода – memory\_profiler и line\_profiler.

**memory\_profiler** — это инструмент для профилирования использования памяти в Python-программах. Он позволяет отслеживать, сколько памяти используется на каждой строке кода, что полезно для выявления утечек памяти, оптимизации использования памяти и понимания, какие части кода потребляют больше всего ресурсов. Результат анализа этой программой представлен на Рисунке 2.1.1.

Изображение выглядит как текст, линия, снимок экрана, число

Автоматически созданное описание

Рисунок 2.1.1 – Результат анализа memory\_profiler

**line\_profiler** — это инструмент для профилирования производительности кода на уровне строк в Python. Он позволяет измерять время выполнения каждой строки кода, что полезно для выявления узких мест в производительности и оптимизации кода. Результат анализа представлен на Рисунке 2.1.2.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1.2 – Результат анализа line\_profiler

### 2.1.1 Внесённые ошибки

В код программы внесены три ошибки:

* в функции вычисления функции Растригина добавлена задержка на 0.3 секунды (Рисунок 2.1.3);
* в функцию вычисления лучшего результата добавлено лишнее заполнение массива (Рисунок 2.1.4);
* что-то там (Рисунок 2.1.5).

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1.3 – Задержка в функции Растригина

Изображение выглядит как текст, линия, снимок экрана, График

Автоматически созданное описание

Рисунок 2.1.6 – Результат анализа memory\_profiler

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1.7 – Результат анализа line\_profiler

## 2.2 Анализ кода на языке C

Для динамического анализа кода на языке C использованы два динамических анализатора – Valgrind и DynamoRIO.

**Valgrind** — это мощный инструмент для динамического анализа программ, написанных на языках C, C++ и других языках, которые могут быть скомпилированы в исполняемые файлы для архитектуры x86. Valgrind предоставляет несколько инструментов для анализа использования памяти, поиска утечек памяти, профилирования производительности и других аспектов качества кода. Результат анализа представлен на Рисунке 2.2.1.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 2.2.1 – Результат анализа Valgrind

**DynamoRIO** — это мощная платформа для динамического инструментирования программ, которая позволяет вставлять инструкции в исполняемый код во время выполнения программы. DynamoRIO поддерживает программы на языках C, C++ и других, которые могут быть скомпилированы в исполняемые файлы для архитектуры x86 и x86-64. Она предоставляет гибкий и мощный механизм для анализа и модификации кода на лету, что делает её полезной для различных задач, таких как профилирование, отладка, оптимизация и анализ безопасности. Результат анализа представлен на Рисунке 2.2.2.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 2.2.2 – Результат анализа DynamoRIO