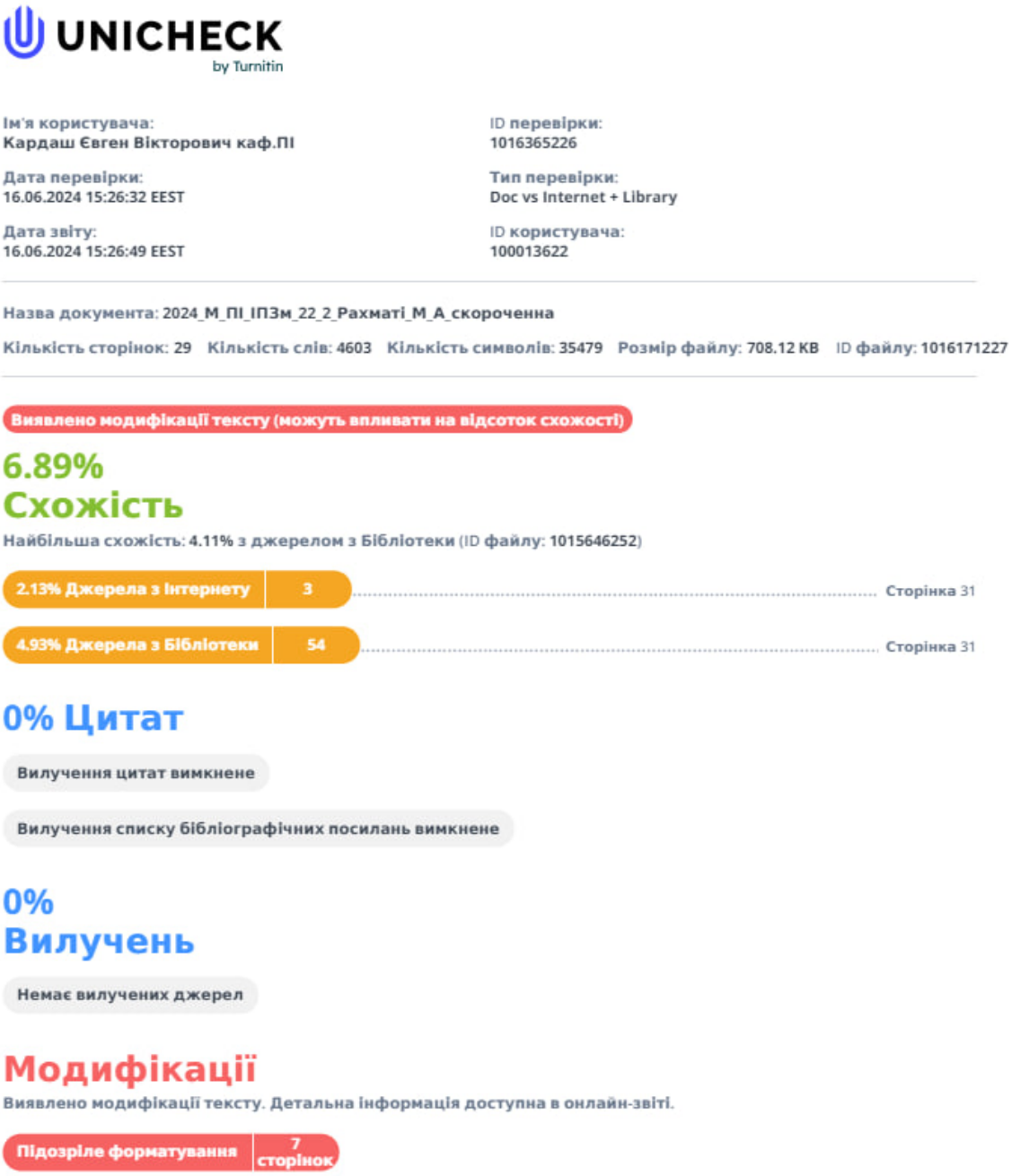
ДОДАТОК А

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Кардаш Євген Вікторович каф.ПІ

ID перевірки:
1016365226

Дата перевірки:
16.06.2024 15:26:32 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
16.06.2024 15:26:49 EEST

ID користувача:
100013622

Назва документа: 2024_М_ПІ_ІПЗм_22_2_Рахматі_М_А_скороченна

Кількість сторінок: 29 Кількість слів: 4603 Кількість символів: 35479 Розмір файлу: 708.12 KB ID файлу: 1016171227

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

6.89%
Схожість

Найбільша схожість: 4.11% з джерелом з Бібліотеки (ID файлу: 1015646252)

2.13% Джерела з Інтернету 3 ..... Сторінка 31

4.93% Джерела з Бібліотеки 54 ..... Сторінка 31

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування 7 сторінок

ДОДАТОК Б

Слайди презентації

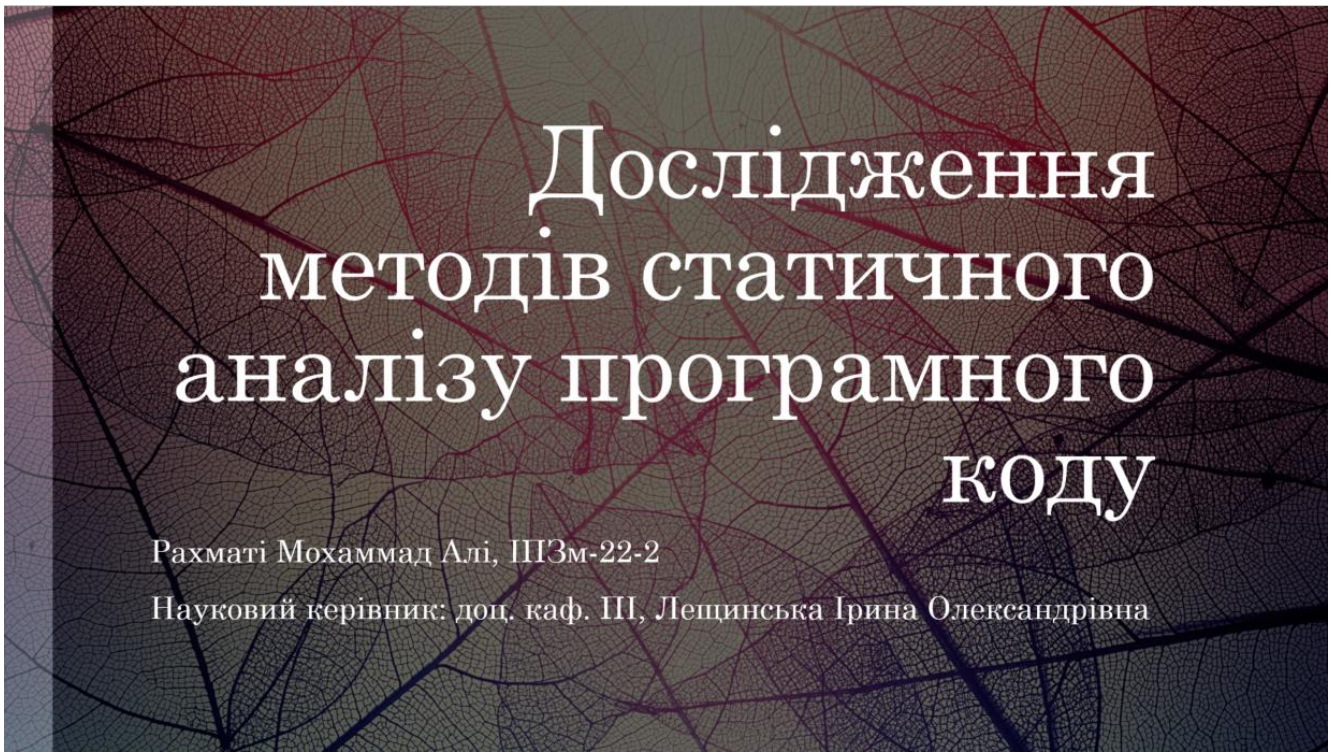


Рисунок Б.1 – Титульний слайд

Рисунок Б.2 – Дослідження

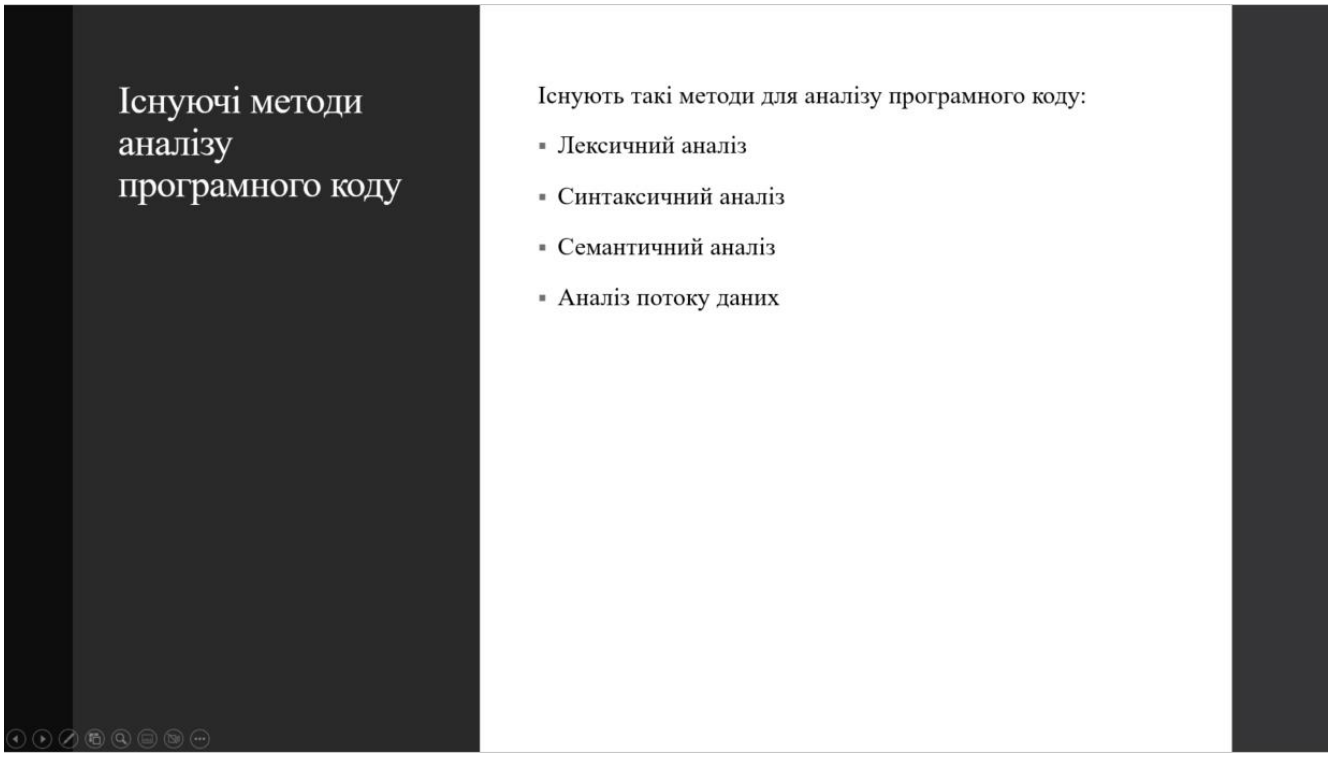


Рисунок Б.3 – Існуючи методи аналізу програмного коду

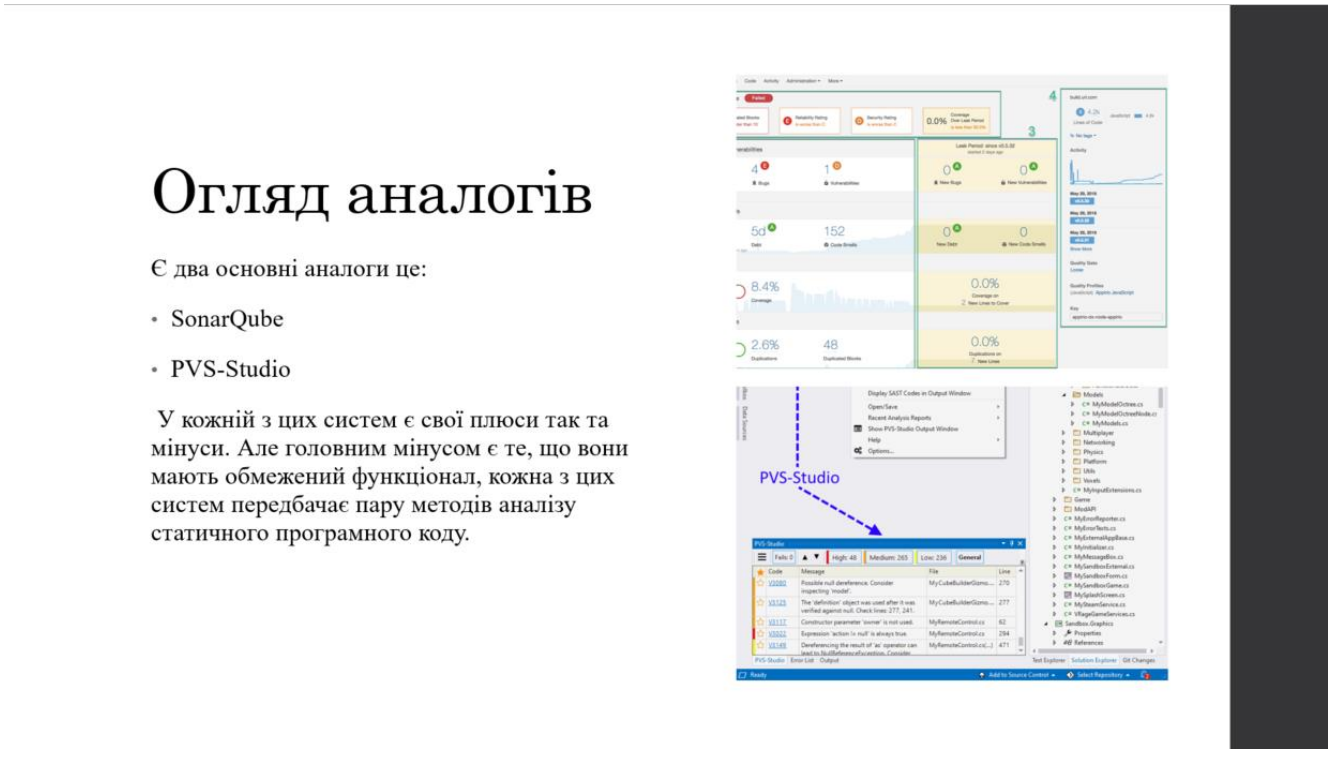


Рисунок Б.4 – Огляд аналогів

Постанова задачі

Провести порівняльний аналіз популярних інструментів статичного аналізу, оцінити їхні можливості та обмеження. Врахувати фактори, такі як підтримка мов програмування, гнучкість конфігурації та можливість інтеграції. Дослідити оптимальні підходи до інтеграції обраних методів статичного аналізу в розробчий процес. Врахувати взаємодію з іншими етапами розробки та вплив на продуктивність розробників.

Рисунок Б.5 – Постановка задачі

Методологія

Основними метриками оцінювання, за допомогою яких здійснюється порівняння трьох інструментів таблиця розбору переваг та недоліків, за якою можна побачити та визначити ефективність інструментів для дослідження. Також побачимо на одному датасетті, які помилки знайшли інструменти.

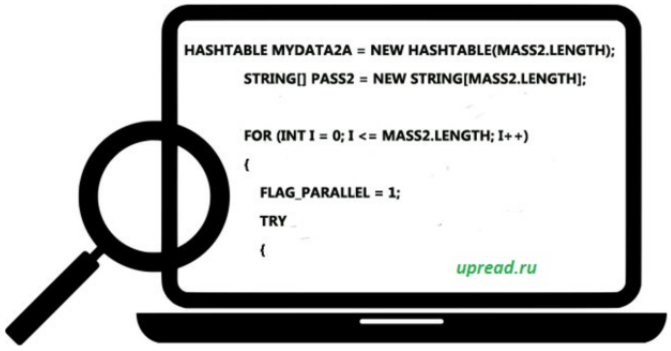
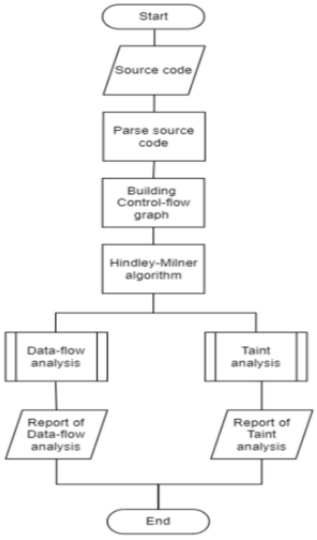


Рисунок Б.6 – Методологія

Архітектура реалізації комбінованого методу статичного аналізу



Для проведення нашого дослідження будемо використовувати архітектуру Хіндлі-Мілнера. Вона допоможе виявляти слабкі місця коду та робить пошук помилок.

Рисунок Б.7 – Архітектура реалізації комбінованого методу статичного аналізу кода

Набір даних для дослідження

Набір даних представляє з себе написаний програмний код, з зробленими навмисно помилками. Так як код буде відрізнятися на мовах програмування, буде наведено приклад на мові Java.

```
// example.js
function addNumbers(a, b) {
    return a + b;
}

function getEvenNumbers(numbers) {
    return numbers.filter(num => num % 2 == 0);
}

let x = 10;
let y = "20";
console.log(addNumbers(x, y));

let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = getEvenNumbers(numbers);
console.log("Even numbers: " + evenNumbers);
```

Рисунок Б.8 – Набір данних для дослідження

Проведення експеременту

```javascript
// example.js
function addNumbers(a, b) {
    return a + b;
}

function getEvenNumbers(numbers) {
    return numbers.filter(num => num % 2 == 0);
}

let x = 10;
let y = "20";
console.log(addNumbers(x, y));

let numbers = [1, 2, 3, 4, 5, 6];
let evenNumbers = getEvenNumbers(numbers);
console.log("Even numbers: " + evenNumbers);
```

```python
from typing import List

def add_numbers(a: int, b: int) -> int:
    return a + b

def get_even_numbers(numbers: List[int]) -> List[int]:
    return [num for num in numbers if num % 2 == 0]

def main() -> None:
    x: int = 10
    y: str = "20"  сва
    result: int = add_numbers(x, y)
    print(f"The sum of {x} and {y} is {result}")

    numbers: List[int] = [1, 2, 3, 4, 5, 6]
    even_numbers: List[int] = get_even_numbers(numbers)
    print(f"Even numbers: {even_numbers}")

if __name__ == "__main__":
    main()
```

```cpp
#include <iostream>
#include <vector>

int addNumbers(int a, int b) {
    return a + b;
}

std::vector<int> getEvenNumbers(const std::vector<int>& numbers) {
    std::vector<int> evens;
    for (int num : numbers) {
        if (num % 2 == 0) {
            evens.push_back(num);
        }
    }
    return evens;
}

int main() {
    int x = 10;
    std::string y = "20";  // Помилка: використання рядка замість числа
    int result = addNumbers(x, std::stoi(y));  // Потенційна помилка
    std::cout << "The sum of " << x << " and " << y << " is " << result << std::endl;

    std::vector<int> numbers = {1, 2, 3, 4, 5, 6};
    std::vector<int> evenNumbers = getEvenNumbers(numbers);
    std::cout << "Even numbers: ";
    for (int num : evenNumbers) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
```

Для проведення експеременту було написано однаковий код на трьох мовах програмування, та у різних середовищах. Було запущенно інструменти для перевірки ESLint для JavaScript, Муру для Python, PVS-Studio для C++.

Рисунок Б.9 – Проведення експеременту

Результат проведеного експеременту для ESLint

```
example.js
  9:17  error  '20' is assigned a value but never used  no-unused-vars
  9:23  error  Strings must use singlequote             quotes
  10:28 error  Expected '===' and instead saw '=='      eqeqeq
  14:19 error  Missing semicolon                        semi

✖ 4 problems (4 errors, 0 warnings)
```

Рисунок Б.10 – Результат проведеного експерименту для ESLint

Результат проведеного експеременту для Муру

```
example.py:10: error: Argument 2 to "add_numbers" has incompatible type "str"; expected "int"
example.py:17: error: Incompatible types in assignment (expression has type "List[int]", variable has
    type "List[str]")
Found 2 errors in 1 file (checked 1 source file)
```

Рисунок Б.11 – Результат проведеного експерименту для Муру

Результат проведеного експеременту для PVS-Studio

```
example.cpp:14: warning: V522: Return of 'stoi(y)' could result in undefined behavior if the string does not represent a valid integer.
example.cpp:18: warning: V519: The 'y' variable is assigned values twice successively. Perhaps this is a mistake. Check lines: 18, 18.
```

Рисунок Б.12 – Результат проведеного експерименту для PVS-Studio

Аналіз отриманих результатів

Результати аналізу показують, що різні інструменти статичного аналізу коду можуть виявляти різні типи помилок та проблем у коді. Зробимо таблицю порівняння ESLint фокусується на стилістичних помилках у JavaScript-коді, Mypy допомагає виявляти помилки типізації у Python-коді, PVS-Studio знаходить потенційні помилки у C++-коді, а SonarQube забезпечує комплексний аналіз для багатьох мов програмування. Використання цих інструментів допомагає забезпечити високу якість коду та зменшити кількість помилок на ранніх етапах розробки.

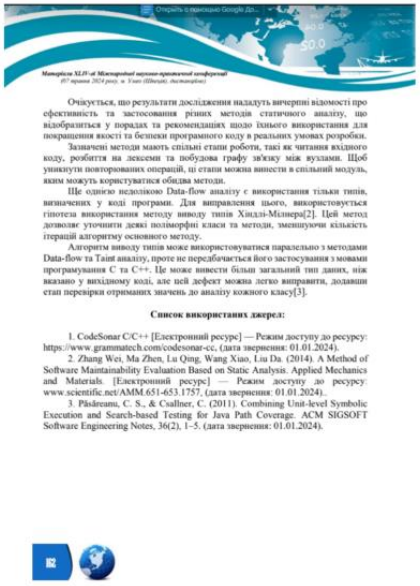
Інструмент	Виявленні помилки
ESLint	- Використання рядка замість числа
	- Використання подвійних лапок замість одинарних
	- Використання "==" замість "==="
	- Відсутня крапка з комою
Mypy	- Неправильний тип аргументу для функції
	- Несумісні типи при присвоєнні значення
PVS-Studio	- Потенційна помилка при конвертації рядка в число
	- Повторне присвоєння змінної

Рисунок Б.13 – Аналіз отриманих результатів

Публікація результатів

Рисунок Б.14 – Апробація

Публікація результатів



Очікується, що результати дослідження нададуть вичерпні відомості про ефективність та застосовання різних методів статичного аналізу, що відобразиться у порадах та рекомендаціях щодо їхнього використання для покращення якості та безпеки програмного коду в реальних умовах розробки.

Зазначені методи мають спільні етапи роботи, такі як читання вхідного коду, розбиття на лексеми та побудова графу зв'язку між вузлами. Щоб уникнути повторюваних операцій, ці етапи можна винести в спільний модуль, яким можуть користуватися обидва методи.

Ще однією недолікою Data-flow аналізу є використання тільки типів, визначених у коді програми. Для виправлення цього, використовується гіпотеза використання методу виводу типів Хіндлі-Мілнера[2]. Цей метод дозволяє уточнити деякі поліморфні класи та методи, зменшуючи кількість ітерацій алгоритму основного методу.

Алгоритм виводу типів може використовуватися паралельно з методами Data-flow та Taint аналізу, проте не передбачається його застосування з мовами програмування C та C++. Це може нивести більш загальний тип даних, ніж вказано у вихідному коді, але цей дефект можна легко виправити, додавши етап перевірки отриманих значень до аналізу кожного класу[3].

Список використаних джерел:

1. CodeSonar C/C++ [Електронний ресурс] — Режим доступу до ресурсу: https://www.grammatech.com/codesonar-cc, (дата звернення: 01.01.2024).
2. Zhang Wei, Ma Zhen, Lu Qing, Wang Xiao, Liu Da. (2014). A Method of Software Maintainability Evaluation Based on Static Analysis. Applied Mechanics and Materials. [Електронний ресурс] — Режим доступу до ресурсу: www.scientific.net/AMM.651-653.1757, (дата звернення: 01.01.2024)..
3. Păsăreanu, C. S., & Csallner, C. (2011). Combining Unit-level Symbolic Execution and Search-based Testing for Java Path Coverage. ACM SIGSOFT Software Engineering Notes, 36(2), 1–5. (дата звернення: 01.01.2024).

Рисунок Б.15 – Апробація

Висновок

Загалом, робота дає уявлення про недооцінювання методів аналізу програмного кода у житті, також про їх потенціал та зручність. Загалом, використання автоматизованих засобів статичного аналізу коду значно підвищує якість та надійність програмного забезпечення. Вони дозволяють виявляти помилки та вразливості на ранніх етапах розробки, що допомагає зменшити витрати на виправлення помилок та забезпечити безпеку програмного забезпечення.

Інтеграція інструментів статичного аналізу в процес розробки дозволяє автоматизувати перевірку коду, забезпечуючи постійний контроль якості на кожному етапі розробки.

Рисунок Б.16 – Висновок

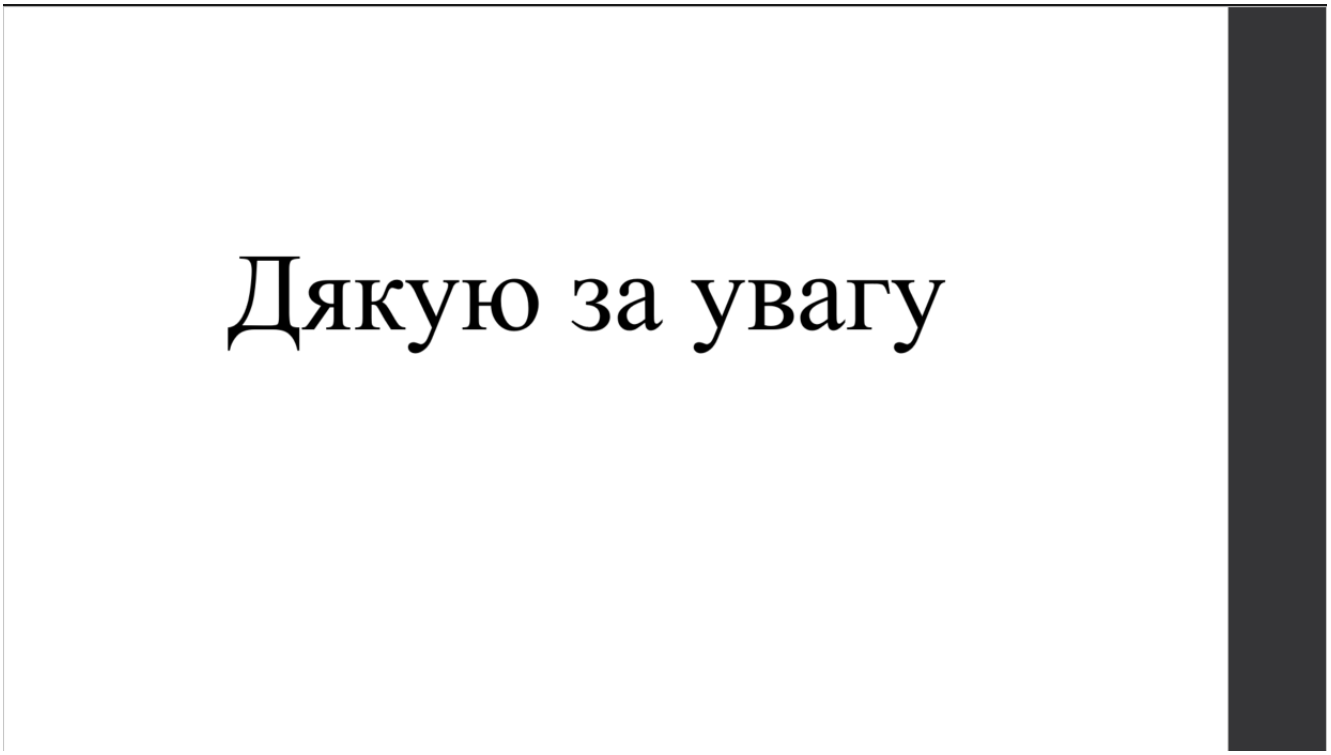


Рисунок Б.17 – Фінальний слайд

ДОДАТОК В

Апробація у вигляді тез у журналі «Сучасні аспекти модернізації науки: стан, проблеми, тенденції розвитку»

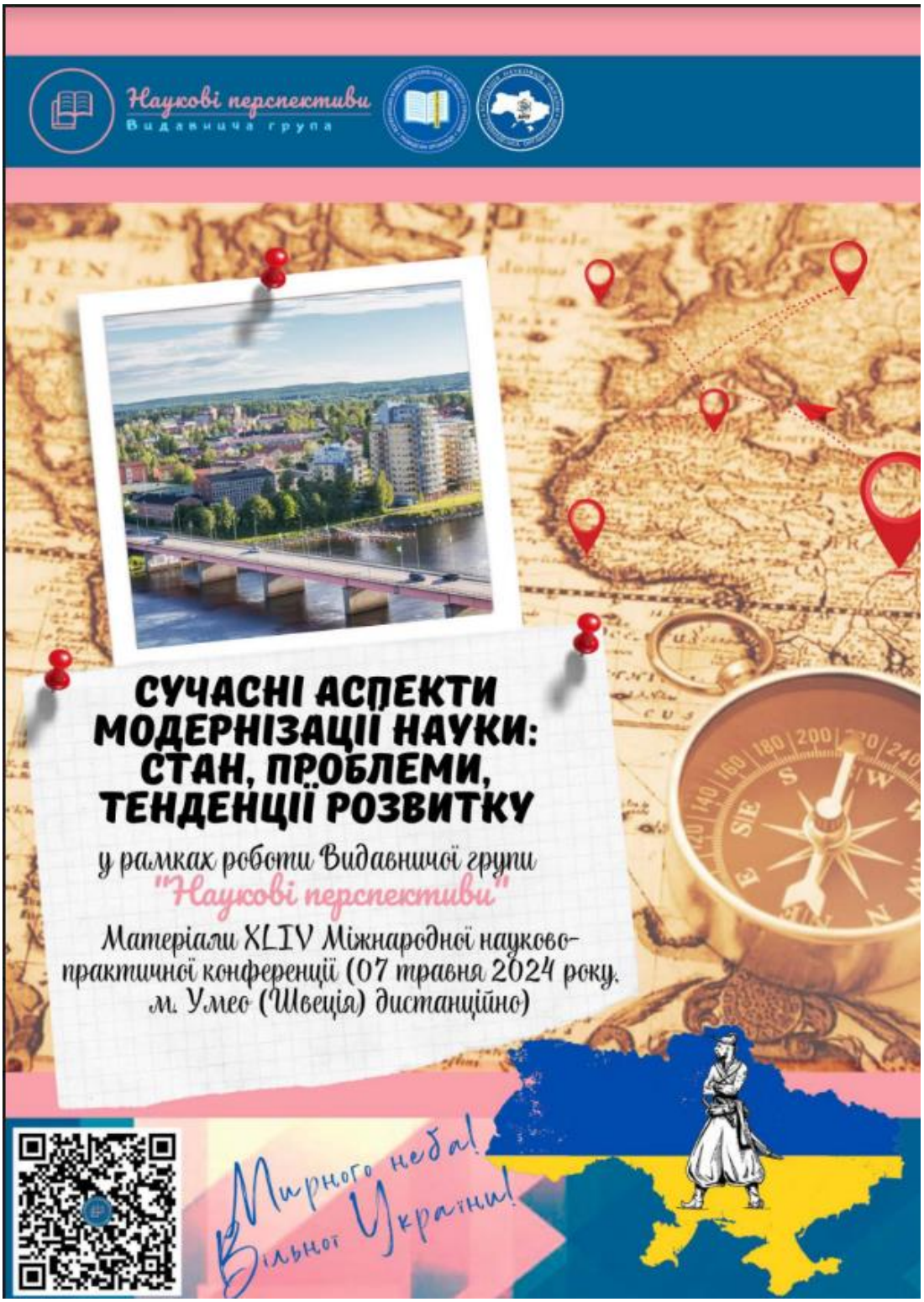


Рисунок В.1 – Титульна сторінка журналу

International Economic Institute s.r.o. (Jesenice, Czech Republic)
Central European Education Institute (Bratislava, Slovakia)
National Institute for Economic Research (Batumi, Georgia)
Al-Farabi Kazakh National University (Kazakhstan)
Institute of Philosophy and Sociology of Azerbaijan National Academy of Sciences (Baku, Azerbaijan)
Batumi Navigation Teaching University (Batumi, Georgia)
Regional Academy of Management (Kazakhstan)
Ukrainian Assembly of Doctors of Sciences in Public Administration (Kyiv, Ukraine)
Association of Scientists of Ukraine (Kyiv, Ukraine)
University of New Technologies (Kyiv, Ukraine)
Interstate Consultants Engineers Guild (Kyiv, Ukraine)
Institute of Education of the Republic of Azerbaijan (Baku, Azerbaijan)

within the Publishing Group "Scientific Perspectives"

MODERN ASPECTS OF MODERNIZATION OF SCIENCE: STATUS, PROBLEMS, DEVELOPMENT TRENDS

Materials of the 44th International Scientific and Practical Conference
May 7, 2024, Umeå (Sweden)

2024

Рисунок В.2 – Титульна сторінка випуску журналу

Рисунок В.3 – Зміст журналу

Рахматі М.А.
Студент кафедри програмної інженерії,
Харківський національний університет радіоелектроніки
м. Харків, Україна

ДОСЛІДЖЕННЯ МЕТОДІВ СТАТИЧНОГО АНАЛІЗУ ПРОГРАМНОГО КОДУ

Щорічний розвиток IT-технологій є невід'ємною частиною нашого інформаційного суспільства, що виявляється у зростанні кількості фахівців, що входять у сферу розробки програмного забезпечення, а також у збільшенні складності самого програмного коду. Сучасні розробники все частіше використовують сторонні бібліотеки та фреймворки, щоб полегшити свою роботу. Однак, незважаючи на ці переваги, неможливо гарантувати відсутність дефектів у використовуваному програмному забезпеченні, що робить систему вразливою.

Інструменти статичного, семантичного та динамічного аналізу є ефективними засобами для виявлення проблем у програмному коді, зокрема щодо якості. Динамічний аналіз ідеально підходить для виявлення помилок, пов'язаних із багато потоковістю та керуванням пам'яттю, хоча й може бути витратним для великих проектів. У той час як статичний аналіз може ефективно використовуватися для виявлення помилок під час процедури перевірки коду, полегшуючи роботу рецензента, він також має свої обмеження, такі як недоліки у точності та можливість видачі помилкових повідомлень.

Однак існує постійне прагнення до удосконалення та оптимізації методів статичного аналізу для підвищення їхньої ефективності, зокрема шляхом прискорення часу виконання. Такі інструменти стають невід'ємними помічниками розробників, забезпечуючи не лише швидке виявлення помилок, але й загальне підвищення продуктивності та надійності розробки програмного забезпечення.

Сучасний ландшафт розробки програмного забезпечення характеризується великим обсягом коду, розподіленими командами розробників та постійним ростом складності проектів. У такому контексті актуальність дослідження методів статичного аналізу програмного коду стає неоспоримою.

Сучасні програмні проекти, включаючи ті, що використовують штучний інтелект, мають величезний обсяг коду. Статичний аналіз дозволяє виявляти потенційні помилки та вразливості на ранніх етапах розробки, сприяючи покращенню структури та якості програм.

Рисунок В.4 – Перша сторінка тез

Розподілена та різноманітна команда розробників стає стандартом. Статичний аналіз є необхідним інструментом для уніфікації стандартів коду та забезпечення єдиної методології в розробці.

У світлі постійно зростаючих загроз кібербезпеки, статичний аналіз дозволяє виявляти та усувати потенційні вразливості, забезпечуючи безпеку програмного забезпечення вже на етапі розробки.

У сфері штучного інтелекту, де точність та надійність є пріоритетами, статичний аналіз допомагає забезпечити стійкість та ефективність роботи алгоритмів, виявляючи потенційні проблеми ще на етапі розробки.

Таким чином, дослідження методів статичного аналізу коду виявляється актуальним та ключовим для забезпечення якості, безпеки та ефективності програмного забезпечення в умовах сучасного інформаційного суспільства.

Запропонований метод базується на підході Data-flow аналізу. Цей метод володіє рядом особливостей, зокрема, він не проводить перевірку коду на існуючі вразливості, на відміну від Taint аналізу. Припускаючи, що об'єднавши обидва ці формальні методи статичного аналізу, можна отримати комбінований метод з покращеною ефективністю через деталізацію результатів аналізатора вихідного коду.

Сучасне середовище розробки програмного забезпечення вимагає вдосконалення методів аналізу коду для забезпечення високої якості, безпеки та ефективності програмних продуктів. У цьому контексті проведення дослідження методів статичного аналізу програмного коду стає критично важливим завданням[1].

Провести глибокий аналіз різноманітних методів статичного аналізу, включаючи статичний аналіз коду, семантичний аналіз, аналіз взаємодії модулів та інші підходи. Зосередитися на їхніх теоретичних основах та практичних застосуваннях.

Визначити переваги та обмеження кожного методу статичного аналізу. Розглянути їхню ефективність в знаходженні помилок, швидкість виконання, витрати ресурсів та інші аспекти.

Провести порівняльний аналіз популярних інструментів статичного аналізу, оцінити їхні можливості та обмеження. Врахувати фактори, такі як підтримка мов програмування, гнучкість конфігурації та можливість інтеграції.

Розглянути, як застосування різних методів статичного аналізу впливає на якість та безпеку програмного коду. Визначити їхні можливості виявлення та усунення вразливостей та помилок.

Дослідити оптимальні підходи до інтеграції обраних методів статичного аналізу в розробчий процес. Врахувати взаємодію з іншими етапами розробки та вплив на продуктивність розробників.

Рисунок В.5 –Друга сторінка тез

Матеріали XLIV-ої Міжнародної науково-практичної конференції
(07 травня 2024 року, м. Умео (Швеція), дистанційно)

Очікується, що результати дослідження нададуть вичерпні відомості про ефективність та застосовання різних методів статичного аналізу, що відобразиться у порадах та рекомендаціях щодо їхнього використання для покращення якості та безпеки програмного коду в реальних умовах розробки.

Зазначені методи мають спільні етапи роботи, такі як читання вхідного коду, розбиття на лексеми та побудова графу зв'язку між вузлами. Щоб уникнути повторюваних операцій, ці етапи можна винести в спільний модуль, яким можуть користуватися обидва методи.

Ще однією недолікою Data-flow аналізу є використання тільки типів, визначених у коді програми. Для виправлення цього, використовується гіпотеза використання методу виводу типів Хіндлі-Мілнера[2]. Цей метод дозволяє уточнити деякі поліморфні класи та методи, зменшуючи кількість ітерацій алгоритму основного методу.

Алгоритм виводу типів може використовуватися паралельно з методами Data-flow та Taint аналізу, проте не передбачається його застосування з мовами програмування C та C++. Це може вивести більш загальний тип даних, ніж вказано у вихідному коді, але цей дефект можна легко виправити, додавши етап перевірки отриманих значень до аналізу кожного класу[3].

Список використаних джерел:

1. CodeSonar C/C++ [Електронний ресурс] — Режим доступу до ресурсу: https://www.grammatech.com/codesonar-cc, (дата звернення: 01.01.2024).

2. Zhang Wei, Ma Zhen, Lu Qing, Wang Xiao, Liu Da. (2014). A Method of Software Maintainability Evaluation Based on Static Analysis. Applied Mechanics and Materials. [Електронний ресурс] — Режим доступу до ресурсу: www.scientific.net/AMM.651-653.1757, (дата звернення: 01.01.2024)..

3. Păsăreanu, C. S., & Csallner, C. (2011). Combining Unit-level Symbolic Execution and Search-based Testing for Java Path Coverage. ACM SIGSOFT Software Engineering Notes, 36(2), 1–5. (дата звернення: 01.01.2024).

Рисунок В.6 – Третя сторінка тез

ДОДАТОК Г

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

1

Експертний висновок результатів перевірки кваліфікаційної роботи

студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-2 (група)

Рахматі М.А.
(прізвище, ім'я, по батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.6 Таблиці	
7.7.2	Якщо подають переліки одного рівня підпорядкованості, на які у звіті немає посилань, то перед кожним із переліків ставлять знак «тире». Якщо у звіті є посилання на переліки, підпорядкованість позначають малими літерами української абетки, далі — арабськими цифрами, далі — через знаки «тире». Після цифри або літери певної позиції переліку ставлять круглу дужку.	27
Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ІІІ протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023.	Рисунок повинен розміщуватися одразу після його згадування у тексті, або на наступній сторінці. Під рисунком повинен бути підпис із словом Рисунок, порядковим номером цього рисунку, через тире з великої літери – назва рисунку та **в круглих дужках вказується джерело з якого взятий цей рисунок, або то, що його виконано самостійно.**	12, далі за текстом.
Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ІІІ протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023.	Назву таблиці друкують з великої літери і розміщують над таблицею з абзацного відступу та **в круглих дужках вказується джерело з якого взята ця таблиця, або то, що вона виконана самостійно. ПРИКЛАД: шаблон, стор.15**	32

Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

18.06.2024