Лабораторна робота №3

**Навчальна дисципліна**: Інженерія надійності програмного забезпечення

**Назва**: Визначення надійності програмного забезпечення на етапі проектування

**Мета**: Оволодіти знаннями про визначення надійності програмного забезпечення на етапі проектування.

**Обладнання**: персональний комп’ютер з встановленою операційною системою Windows, або Linux.

**Завдання**:

1. Вивчити математичну модель прогнозування надійності програмного забезпечення на етапі проектування на основі моделі Холстеда.

2. Отримати практичні навички розрахунку проектної надійності програмного забезпечення.

Виконав студент 544 групи

Максимович Микола Юрійович

**Висновки**

Результатом лабораторної роботи є python скрипт для оцінки кількості помилок в коді. Скрипт дозволяє вибрати 1 та більше вхідних файлів які будуть аналізуватись. Програма вміє аналізувати код мови python та java. Після аналізу файлів програма проводить обчислення кількості помилок з використанням методу Холстеда та виводить результат в консоль.

**Додатки**

**Додаток 1 – Лістинг коду, файл “main.py”**

from tkinter import filedialog

import math

import ast

import javalang

EXTENSIONS = ['.py', '.java']

def chooseFiles():

    chosen\_file\_paths = filedialog.askopenfilenames(title='Виберіть файли програм які містять програмний код',

                                                  filetypes=[('файли з кодом', EXTENSIONS)])

    if chosen\_file\_paths:

        print("Ви вибрали:", chosen\_file\_paths)

        return chosen\_file\_paths

    else:

        print("Ви нічого не вибрали.")

def parse\_py\_file(file):

    code = file.read()

    tree = ast.parse(code)

    dict\_constructs = 0

    subroutines = 0

    variable\_arrays = 0

    local\_labels = 0

    constants = 0

    for node in ast.walk(tree):

        if isinstance(node, ast.FunctionDef):

            subroutines += 1

        if isinstance(node, ast.Dict) or isinstance(node, ast.Set):

            dict\_constructs += 1

        if isinstance(node, ast.List) or isinstance(node, ast.Tuple):

            variable\_arrays += 1

        if isinstance(node, ast.Name) and not isinstance(node.ctx, ast.Load):

            local\_labels += 1

        if isinstance(node, ast.Constant):

            constants += 1

    return {

        "dict\_constructs": dict\_constructs,

        "subroutines": subroutines,

        "variable\_arrays": variable\_arrays,

        "local\_labels": local\_labels,

        "constants": constants

    }

def parse\_java\_file(file):

    code = file.read()

    tree = javalang.parse.parse(code)

    dict\_constructs = 0

    subroutines = 0

    variable\_arrays = 0

    local\_labels = 0

    constants = 0

    for path, node in tree:

        if isinstance(node, javalang.tree.CompilationUnit):

            subroutines += 1

        if isinstance(node, javalang.tree.DictionaryInitializer):

            dict\_constructs += 1

        if isinstance(node, javalang.tree.ArrayInitializer):

            variable\_arrays += 1

        if isinstance(node, javalang.tree.Statement):

            if 'break' in str(node):

                local\_labels += 1

        if isinstance(node, javalang.tree.ConstantDeclaration):

            constants += 1

    return {

        "dict\_constructs": dict\_constructs,

        "subroutines": subroutines,

        "variable\_arrays": variable\_arrays,

        "local\_labels": local\_labels,

        "constants": constants

    }

def calculate\_error(input: dict[str,int]) -> float:

    n\_sk = input["dict\_constructs"] # кількість використовуваних словникових конструкцій

    n\_pp = input["subroutines"] # кількість підпрограм

    n\_mp = input["variable\_arrays"] # кількість масивів змінних

    n\_met = input["local\_labels"] # кількість локальних міток

    n\_k = input["constants"] # кількість констант

    n1 = n\_sk + n\_pp # число операцій

    n2 = n\_mp + n\_met + n\_k # число операндів

    n = n1 + n2 # словник

    a = n1 \* math.log2(n1) + n2 \* math.log2(n2) # теоретична довжина програми

    v = a \* math.log2(n) # обсяг програми, що спостерігається

    vy = 3000 # питомий обсяг програми, рівний середньому обсягу програми, що припадає на один дефект

    return v / vy

def concat(x1,x2:dict[str, int]):

    if len(x1) == 0:

        x1 = x2

    else:

        for key in x2.keys():

            x1[key] += x2[key]

    return x1

def main():

    chosen\_file\_paths = chooseFiles()

    if chosen\_file\_paths is None:

        exit(0)

    parsed\_data = {}

    for chosen\_file\_path in chosen\_file\_paths:

        with open(chosen\_file\_path, "r") as file:

            if chosen\_file\_path.split(".")[-1] == "py":

                parsed\_data = concat(parsed\_data, parse\_py\_file(file))

            elif chosen\_file\_path.split(".")[-1] == "java":

                parsed\_data = concat(parsed\_data, parse\_java\_file(file))

    if parsed\_data:

        print("Число дефектів в програмі: ", calculate\_error(parsed\_data))

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**Додаток 2 – скріншоти результату виконання додатку**

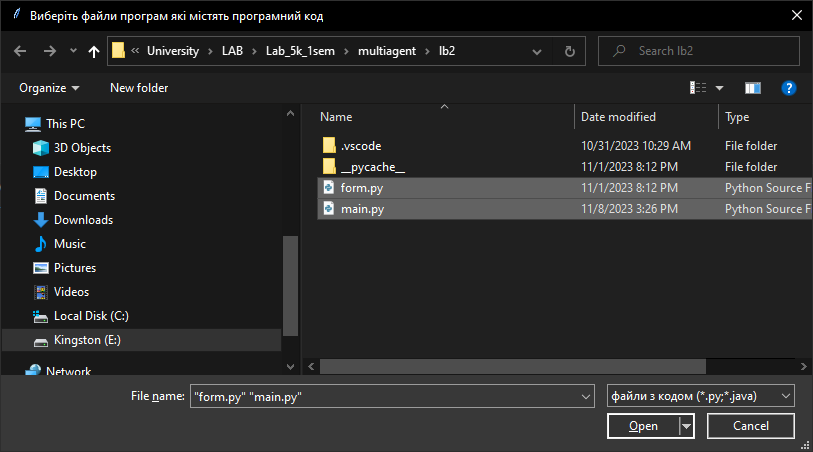


Рисунок 1 – додаток пропонує вибрати файли з кодом

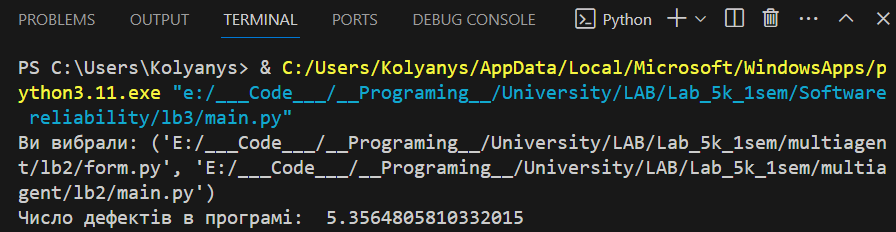


Рисунок 2 – вивід в консоль шлях до файлів, які вибрав користувач, та вивід числа дефектів в останньому рядку.

За результатами обчислень додатку, кількість дефектів дорівнює близько 5. Це може бути правдою, адже я сам знаю про існування двох, не критичних, помилок в додатку який перевірявся.

**Контрольні запитання**

1. Особливості програмного забезпечення як об'єкта аналізу надійності включають складність, динамічність та залежність від вхідних даних.
2. Залежності зміни надійності програмних та апаратних засобів ІС від часу можуть бути різними і залежать від конкретного виробника та середовища використання.
3. Надійність програмного забезпечення за стандартом ДСТУ ISO/ІЄС 9126-1:2013 - це властивість програмного забезпечення, яка забезпечує його здатність виконувати функції без відмов та забезпечувати задоволення користувачів за встановленими умовами.
4. Основні причини помилок програмного забезпечення включають програмні баги, недбалість розробників, невідповідність вимогам та зміни в середовищі використання.
5. Джерела помилок (загрози надійності) програмного забезпечення можуть бути внутрішніми (програмні помилки, неправильний дизайн) і зовнішніми (атаки, збої апаратного забезпечення, зміни в середовищі).
6. Відмінності програмних та апаратних відмов полягають у тому, що програмні відмови пов'язані із неправильною роботою програмного коду, тоді як апаратні відмови виникають через проблеми з апаратними компонентами, такими як процесори, пам'ять тощо.