МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Кафедра математического обеспечения и применения ЭВМ

**Отчет по лабораторной работе №2**

на тему:

«**Определение метрических характеристик программного обеспечения**»

по дисциплине

«**Качество и тестирование программного обеспечения**»

модуль

**«Метрология программного обеспечения»**

Вариант 7

Выполнили:

Студенты группы

КТбо2-8

Жалнин Д. И.

Ледерер П. А.

Кочубей Д. С.

Проверил:

Старший преподаватель

Кафедры МОП ЭВМ

Проскуряков А. В.

Оценка

вы попались как дешевки, пойманы за руку как грязные дешевки

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Таганрог 2021

Содержание

[1 Цель работы 3](#_Toc72772965)

[2 Задание 3](#_Toc72772966)

[3 Ход работы 3](#_Toc72772967)

[3.1 Теоретическая часть 3](#_Toc72772968)

[3.2 Практическая часть 4](#_Toc72772969)

[3.2.1 Разработанная программа 4](#_Toc72772970)

[3.2.2 Вычисление объема программы 6](#_Toc72772971)

[4 Вывод 8](#_Toc72772972)

[5 Список используемой литературы 9](#_Toc72772973)

[Приложение А 10](#_Toc72772974)

1. Цель работы

Целью данной работы является знакомство с метрическими характеристиками программного обеспечения, а также их определение посредством использования метрик. Made by Ruslan and Misha.

1. Задание

Составить программу реализации арифметического выражения согласно варианту 7.

Подсчитать количество операторов и операндов и определить объем программы.

Написать программу-анализатор, которая определит объем исходной программы.

Составьте отчет по результатам выполнения лабораторно-практической работы.

1. Ход работы
   1. Теоретическая часть

Метрика – отношение, определяющее расстояние между элементами множества. Введение метрики весьма важно, поскольку позволяет говорить о формализации оценки разработки ПО.

В исследовании метрик ПО различают 2 основных направления:

1. Поиск метрик, характеризующих наиболее специфические свойства программ, т.е. метрик оценки самого ПО;
2. Использование метрик для оценки технических характеристик и факторов разработки программ, т.е. метрик оценки условий разработки программ.

Качество программного обеспечения неразрывно связано с понятием измерения характеристик программ и программных систем.

При выборе метрик, характеризующих основные направления, принимались во внимание требования: трудоемкость ПО, принимались оценки качества воспроизводимости результатов измерений, вычислений, адекватность обобщенному интуитивному представлению программистов о качестве программы. Исходя из этого, акцент сделан на оценках качества программ, выраженных аналитическими зависимостями, что позволяет говорить о возможной автоматизации их расчета.

Обычно выделяют 3 группы метрик сложности ПО:

* + Метрики размера программ;
  + Метрики сложности потока управления программ;
  + Метрики сложности потока данных программ.

К группе оценок размера программ можно отнести также и метрику Холстеда. За базу данной метрики принят подсчет количества операторов и операндов, используемых в программе, для вычисления размера программы.

* 1. Практическая часть
     1. Разработанная программа

В ходе лабораторной работы была создана утилита, вычисляющая объем программного обеспечения, написанного на языке Java (см. Рисунок 1). Made by

Изображение выглядит как текст

Автоматически созданное описание

Рисунок — Разработанная программа

Данная утилита написана при помощи языка программирования Java с использованием библиотеки «javaparser-core-3.22.1». Анализ файла и подсчет количества операторов и операндов реализован с помощью подключаемой библиотеки «javaparser» по следующему алгоритму:

1. Происходит «парсинг» кода программы на части, которые вносятся в массив – каждой части соответствует некоторый сегмент кода, некоторые части могут включать в себя другие части.
2. В результате прохождения массиву выбираются именно те части, которые содержат в себе операнд или оператор.
3. Найденные части кода вносятся в один из «справочников» со значением 1. Ключом справочника является строка, значением – целое число. Если в справочнике уже найден соответствующий ключ, значение по этому ключу увеличивается на 1.
4. В результате работы программы происходит вывод двух словарей в виде таблиц, подсчитывается количество уникальных операторов и операндов, их общее количество, происходит расчет и вывод объема программы по заданной формуле.

Также в результате работы была разработана программа на языке Java для решения математического выражения, приведенная на Листинге 1.

Листинг 1 — Решение арифметического выражения на языке Java

import java.util.Scanner;

import static java.lang.Math.cos;

import static java.lang.Math.pow;

import static java.lang.Math.sqrt;

import static java.lang.Math.tan;

import static java.lang.System.in;

import static java.lang.System.out;

public class Calc {

public static void main(String[] args) {

double x,y,z,c;

Scanner scanner = new Scanner(in);

out.println("Enter X: >> ");

x = scanner.nextDouble();

out.println("Enter Y: >> ");

y = scanner.nextDouble();

out.println("Enter Z: >> ");

z = scanner.nextDouble();

c = log(2\*x)-sqrt(6\*atan(z));

out.println("Result >> " + c);

}

}}

* + 1. Вычисление объема программы

Для вычисления объема программы используется метрика Холстеда, в которой n — число простых операторов и операндов, а N — общее количество использованных операторов и операндов. Данная метрика выражается следующей формулой:

Подсчет количества операторов и операндов без использования программы можно увидеть в Таблице 1 и Таблице 2.

Таблица — Количество операторов, подсчитанное вручную

|  |  |  |
| --- | --- | --- |
| Оператор | j | f1j |
| atan | 1 | 1 |
| () | 2 | 12 |
| {} | 3 | 2 |
| log | 4 | 1 |
| sqrt | 5 | 1 |
| scanner.nextDouble() | 6 | 3 |
| \* | 7 | 2 |
| + | 8 | 1 |
| ; | 9 | 19 |
| = | 10 | 4 |
| - | 11 | 1 |
| out.println | 12 | 4 |
| Всего: | 12 | 51 |

Таблица — Количество операндов, подсчитанное вручную

|  |  |  |
| --- | --- | --- |
| Операнды | j | f2j |
| "Enter Y: >> " | 1 | 1 |
| c | 2 | 3 |
| in | 3 | 1 |
| "Enter Z: >> " | 4 | 1 |
| "Enter X: >> " | 5 | 1 |
| "Result >> " | 6 | 1 |
| out | 7 | 4 |
| 2 | 8 | 1 |
| 6 | 9 | 1 |
| x | 10 | 3 |
| scanner | 11 | 4 |
| y | 12 | 2 |
| z | 13 | 3 |
| Всего | 13 | 26 |

На основе вышеприведенных ручных подсчетов, был определен объем программы:

Подсчет количества операторов и операндов c использованием программы можно увидеть в Таблице 3 и Таблице 4.

Таблица — Количество операторов, подсчитанное программой

|  |  |  |
| --- | --- | --- |
| Оператор | j | f1j |
| atan | 1 | 1 |
| () | 2 | 12 |
| {} | 3 | 2 |
| log | 4 | 1 |
| sqrt | 5 | 1 |
| scanner.nextDouble() | 6 | 3 |
| \* | 7 | 2 |
| + | 8 | 1 |
| ; | 9 | 19 |
| = | 10 | 4 |
| - | 11 | 1 |
| out.println | 12 | 4 |
| Всего: | 12 | 51 |

Таблица — Количество операндов, подсчитанное программой

|  |  |  |
| --- | --- | --- |
| Операнды | j | f2j |
| "Enter Y: >> " | 1 | 1 |
| c | 2 | 3 |
| in | 3 | 1 |
| "Enter Z: >> " | 4 | 1 |
| "Enter X: >> " | 5 | 1 |
| "Result >> " | 6 | 1 |
| out | 7 | 4 |
| 2 | 8 | 1 |
| 6 | 9 | 1 |
| x | 10 | 3 |
| scanner | 11 | 4 |
| y | 12 | 2 |
| z | 13 | 3 |
| Всего | 13 | 26 |

На основе вышеприведенных подсчетов написанной утилиты, был определен объем программы:

Вывод программы приведен на Рисунке 2.

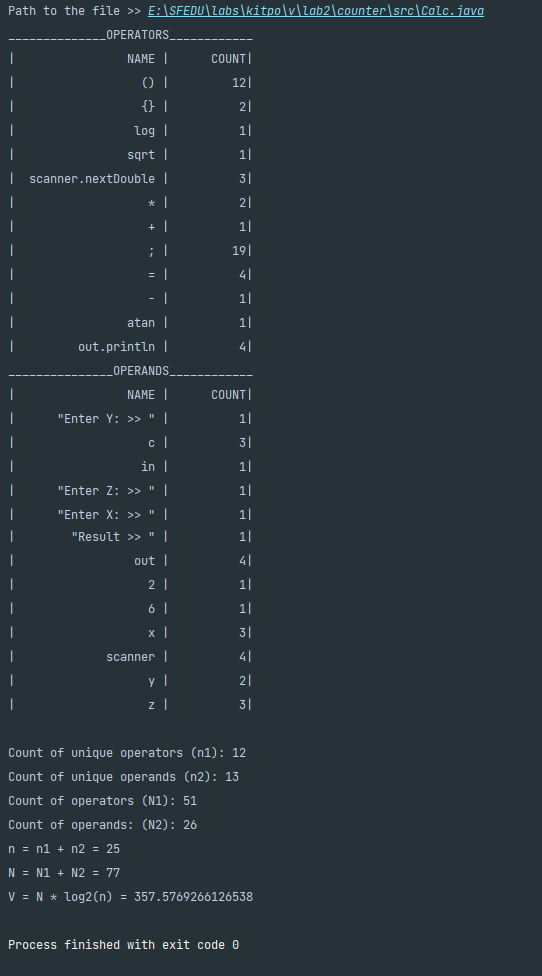


Рисунок — Вывод разработанной программы

Из равенства ручного и программного вычисления можно сделать вывод, что написанная утилита командной строки корректно считает объем программ на языке Java.

1. Вывод

В ходе данной лабораторной работы было осуществлено знакомство с метрическими характеристиками программного обеспечения, а также их определение посредством использования метрик. Была написана программа, позволяющая посчитать объем программного кода, вычисленные значения которой не уступают посчитанным вручную.

1. Список используемой литературы
2. Репозиторий библиотеки «javaparser» [Электронный ресурс] URL: [https://github.com/javaparser/javaparser](https://github.com/javaparser/javaparser%20)
3. Программный код и его метрики [Электронный ресурс] URL: <https://habr.com/ru/company/intel/blog/106082/>

Приложение А

1. Calc.java

import java.util.Scanner;

import static java.lang.Math.atan;

import static java.lang.Math.log;

import static java.lang.Math.sqrt;

import static java.lang.System.in;

import static java.lang.System.out;

public class Calc {

public static void main(String[] args) {

double x,y,z,c;

Scanner scanner = new Scanner(in);

out.println("Enter X: >> ");

x = scanner.nextDouble();

out.println("Enter Y: >> ");

y = scanner.nextDouble();

out.println("Enter Z: >> ");

z = scanner.nextDouble();

c = log(2\*x)-sqrt(6\*atan(z));

out.println("Result >> " + c);

}

}

1. Metrics.java

import com.github.javaparser.JavaParser;

import com.github.javaparser.ast.body.VariableDeclarator;

import com.github.javaparser.ast.expr.AssignExpr;

import com.github.javaparser.ast.expr.BinaryExpr;

import com.github.javaparser.ast.expr.IntegerLiteralExpr;

import com.github.javaparser.ast.expr.MethodCallExpr;

import com.github.javaparser.ast.expr.NameExpr;

import com.github.javaparser.ast.expr.StringLiteralExpr;

import com.github.javaparser.ast.expr.UnaryExpr;

import java.io.File;

import java.io.IOException;

import java.nio.file.Files;

import java.util.HashMap;

import java.util.Scanner;

public class Metrics {

static HashMap<String, Integer> operators = new HashMap<>();

static HashMap<String, Integer> operands = new HashMap<>();

public static void main(String[] args) throws IOException {

String path;

Scanner scanner = new Scanner(System.in);

if (args.length != 0 && !args[0].isEmpty() && !args[0].isBlank()) {

path = args[0];

} else {

System.out.print("Path to the file >> ");

path = scanner.nextLine();

}

File file = new File(path);

if (!file.isFile() || !file.exists() || !file.canRead()) {

System.out.println("Cannot read the file.");

return;

}

String fileString = Files.readString(file.toPath());

JavaParser parser = new JavaParser();

parser.parse(fileString).getResult().get()

.walk(node -> {

if (node instanceof UnaryExpr) {

UnaryExpr e = (UnaryExpr) node;

putOperator(e.getOperator().asString());

} else if (node instanceof AssignExpr) {

AssignExpr e = (AssignExpr) node;

putOperator(e.getOperator().asString());

} else if (node instanceof BinaryExpr) {

BinaryExpr e = (BinaryExpr) node;

putOperator(e.getOperator().asString());

} else if (node instanceof VariableDeclarator) {

putOperand(node.toString().split(" ")[0]);

} else if (node instanceof StringLiteralExpr) {

putOperand(node.toString());

} else if (node instanceof NameExpr) {

putOperand(node.toString());

} else if (node instanceof MethodCallExpr) {

MethodCallExpr e = (MethodCallExpr) node;

putOperator(e.toString().split("\\(")[0]);

} else if (node instanceof IntegerLiteralExpr) {

putOperand(node.toString());

}

});

for (char c : fileString.toCharArray()) {

if (c == '(') {

putOperator("()");

}

if (c == '{') {

putOperator("{}");

}

if (c == ';') {

putOperator(";");

}

}

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_OPERATORS\_\_\_\_\_\_\_\_\_\_\_\_");

print(operators);

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_OPERANDS\_\_\_\_\_\_\_\_\_\_\_\_");

print(operands);

System.out.println();

System.out.println("Count of unique operators (n1): " + operators.size());

System.out.println("Count of unique operands (n2): " + operands.size());

int cOperators = 0;

int cOperands = 0;

for (Integer value : operators.values()) {

cOperators += value;

}

for (Integer value : operands.values()) {

cOperands += value;

}

System.out.println("Count of operators (N1): " + cOperators);

System.out.println("Count of operands: (N2): " + cOperands);

int N = cOperands + cOperators;

int n = operators.size() + operands.size();

System.out.println("n = n1 + n2 = " + n);

System.out.println("N = N1 + N2 = " + N);

System.out.println("V = N \* log2(n) = " + N \* (Math.log(n) / Math.log(2)));

}

private static void print(HashMap<String, Integer> operators) {

System.out.printf("|%20s | %10s|\n", "NAME", "COUNT");

operators.forEach((k, v) -> {

System.out.printf("|%20s | %10s|\n", k, v);

});

}

private static void putOperator(String str) {

if (operators.containsKey(str)) {

operators.put(str, operators.get(str) + 1);

} else {

operators.put(str, 1);

}

}

private static void putOperand(String str) {

if (operands.containsKey(str)) {

operands.put(str, operands.get(str) + 1);

} else {

operands.put(str, 1);

}

}

}