МИНИСТЕРСТВО ОБРАЗОВАНИЕ И НАУКИ УКРАИНЫ

НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ХАРЬКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ»

Кафедра стратегического управления

Отчет к лабораторной работе №1

«Изучение метрик качества программного обеспечения»

Вариант 10

Выполнил:

студент группы КН-26

Горовенко А.В.

Проверил:

Лисицкий В.Л.

Харьков, 2018

Тема: изучение метрик качества программного обеспечения

Цель: составить алгоритм и программу, которые находят минимальное число слагаемых в сумме членов ряда, при котором эта сумма станет больше 120, и вывести найденную сумму, слагаемое и его номер.

1 + 1\*2 + 1\*2\*3 + 1\*2\*3\*4 +…

Основные теоретические сведение.

Ка́чество програ́ммного обеспечения — способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям (ISO/IEC 25000:2014).

Другие определения из стандартов:

* весь объём признаков и характеристик программ, который относится к их способности удовлетворять установленным или предполагаемым потребностям (ГОСТ Р ИСО/МЭК 9126-93, ISO 8402:94).
* степень, в которой система, компонент или процесс удовлетворяют потребностям или ожиданиям заказчика, или пользователя (IEEE Std 610.12-1990).

Метрика — это метод, используемый для численного выражения качества или же достижения поставленных целей в каком-либо процессе.

В процессе разработки программного обеспечения метрики используются:

* Для контроля над процессом разработки в целом, а в тестировании в частности
* Для оценки прогресса выполнения работ по тестированию ПО
* Для последующего планирования процессов.

Также к группе метрик, основанных на подсчете некоторых единиц в коде программы, относят метрики Холстеда [3]. Данные метрики основаны на следующих показателях:

* n1 — число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов),
* n2 — число уникальных операндов программы (словарь операндов),
* N1 — общее число операторов в программе,
* N2 — общее число операндов в программе,
* n1' — теоретическое число уникальных операторов,
* n2' — теоретическое число уникальных операндов.
* Учитывая введенные обозначения, можно определить:
* n=n1+n2 — словарь программы,
* N=N1+N2 — длина программы,
* n'=n1'+n2' — теоретический словарь программы,
* N'= n1\*log2(n1) + n2\*log2(n2) — теоретическая длина программы (для стилистически корректных программ отклонение N от N' не превышает 10%)
* V=N\*log2n — объем программы,
* V'=N'\*log2n' — теоретический объем программы, где n\* — теоретический словарь программы.
* L=V'/V — уровень качества программирования, для идеальной программы L=1
* L'= (2 n2)/ (n1\*N2) — уровень качества программирования, основанный лишь на параметрах реальной программы без учета теоретических параметров,
* EC=V/(L')2 — сложность понимания программы,
* D=1/ L' — трудоемкость кодирования программы,
* y' = V/ D2 — уровень языка выражения
* I=V/D — информационное содержание программы, данная характеристика позволяет определить умственные затраты на создание программы
* E=N' \* log2(n/L) — оценка необходимых интеллектуальных усилий при разработке программы, характеризующая число требуемых элементарных решений при написании программы

При применении метрик Холстеда частично компенсируются недостатки, связанные с возможностью записи одной и той же функциональности разным количеством строк и операторов.

Самой распространенной оценкой, основанной на анализе получившегося графа, является цикломатическая сложность программы (цикломатическое число Мак-Кейба) [4]. Она определяется как M = m – n + 2p, где m — количество дуг, n — количество вершин, p — число компонент связности. Число компонентов связности графа можно рассматривать как количество дуг, которые необходимо добавить для преобразования графа в сильно связный. Сильно связным называется граф, любые две вершины которого взаимно достижимы. Для графов корректных программ, т. е. графов, не имеющих недостижимых от точки входа участков и «висячих» точек входа и выхода, сильно связный граф, как правило, получается путем замыкания дугой вершины, обозначающей конец программы, на вершину, обозначающую точку входа в эту программу. По сути M определяет число линейно независимых контуров в сильно связном графе. Так что в корректно написанных программах p=1, и поэтому формула для расчета цикломатической сложности приобретает вид:

M = m – n + 2

К сожалению, данная оценка не способна различать циклические и условные конструкции. Еще одним существенным недостатком подобного подхода является то, что программы, представленные одними и теми же графами, могут иметь совершенно разные по сложности предикаты (предикат — логическое выражение, содержащее хотя бы одну переменную).

Для исправления данного недостатка Г. Майерсом была разработана новая методика. В качестве оценки он предложил взять интервал (эта оценка еще называется интервальной) M = u + 1 , где u для простых предикатов равно нулю, а для n-местных M=n-1. Данный метод позволяет различать разные по сложности предикаты, однако на практике он почти не применяется.

Ход работы:

1. Разработать алгоритм решения для заданной задачи.

Изобразим алгоритм решений в виде диаграммы деятельности, использования и классов, которые показаны на рисунках 1.1 – 1.3 соответственно.

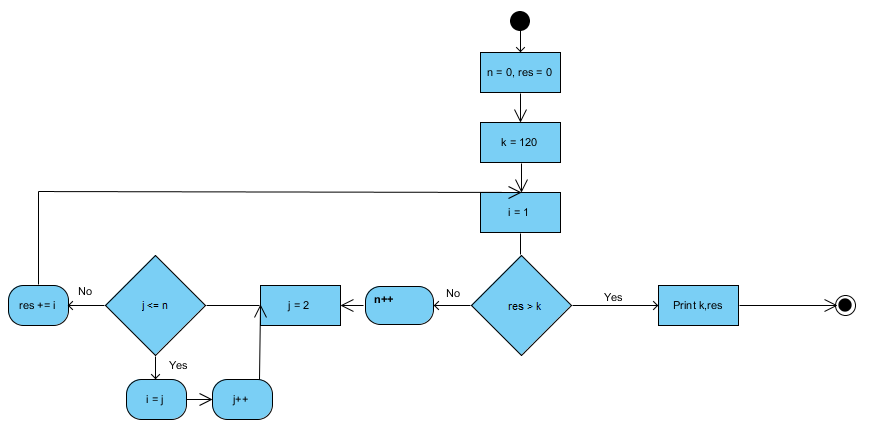


Рисунок 1.1 – Диаграмма деятельности

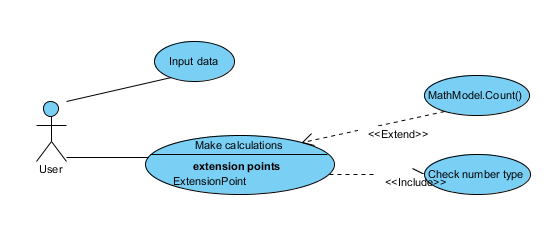


Рисунок 1.2 – Диаграмма использования

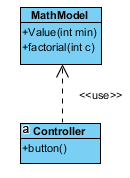


Рисунок 1.3 – Диаграмма классов

1. Написать программу для решение поставленной задачи.

Код программы написанный на c#:

private void button\_Click(object sender, RoutedEventArgs e)

{

try

{

int res = 0, last = 0;

int n = 0;

int min = int.Parse(minTextBox.Text);

while (res < min)

{

n++;

last = factorial(n);

res += last;

}

result.Content = $"n = {n}, last = {last}";

}

catch(Exception x)

{

result.Content = "Некоректный ввод";

}

}

private static int factorial(int c)

{

if (c == 1)

return 1;

return c \* factorial(c - 1);

}

}

Код программы написанный на Java:

**public static int**[] value(**int** min){  
 **int** result = 0, last = 0;  
 **int** n = 0;  
 **while**(result < min){  
 n++;  
 last = *factorial*(n);  
 result += last;  
 }  
 **return new int**[] {n, last};  
 }  
 **private static int** factorial(**int** c){  
 **if**(c == 1)  
 **return** 1;  
 **return** c\**factorial*(c-1);  
 }

1. Метрики

Метрика Холстеда:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  | *n* | *N* | *\** |
| C# | 18 | 18 | 51 | 37 | 36 | 88 | 5 |
| Java | 19 | 11 | 39 | 29 | 30 | 68 | 5 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | *V* | *V\** | *L* |  |  | *E* | *T* | S |
| C# | 150,11 | 454,95 | 19,651 | 0,043 | 0,848 | 24,592 | 10532,66 | 752,33 | 14 |
| Java | 118,764 | 333,66 | 19,651 | 0,058 | 1,157 | 13,322 | 5665,46 | 404,67 | 14 |

Метрика Мак-Кейба:

M = m – n + 2

m – количество ребер

n – количество вершин

M = 12 - 11 + 2 = 3

M = u + 1

u – количество операторов условий

M = 2 + 1 = 3

Из всех возможных предупреждений в данном программном коде присутствует предупреждение вида: .

4)Результат выполнение программы.

На рисунках 4.1 – 4.2 показан результат выполнение программ на языках Java и C# соответственно.

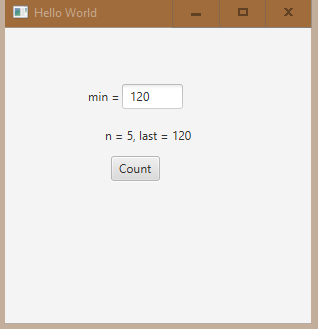


Рисунок 4.1 – выполнение программы на языке Java

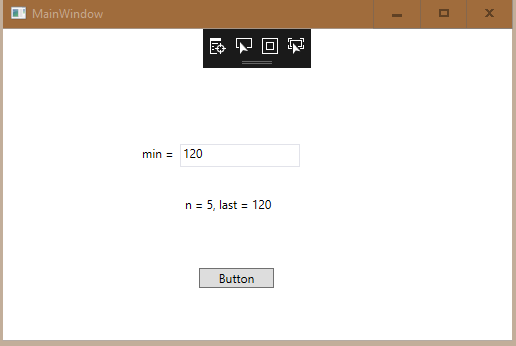


Рисунок 4.2 – выполнение программы на языке C#

Выводы: в ходе выполнение лабораторной работы было созданы диаграммы, а имеено диаграммы деятельности, использование и классов. По данным диаграмам был написан код на языках C# и Java, а также рассмотрены метрики Холлстеда и МакКейба.