**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования «Уральский федеральный университет

имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий - РТФ

Центр ускоренного обучения

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**

по дисциплине «Конструирование программного обеспечения»

**Тема:** метрики Холстеда

Студент гр. РИВ-300027у Я.В.Козлов Л.А.Кайгородова

О.В.Дрон

Д.И.Кудинов

В.И.Пинтак

|  |
| --- |
|  |

Преподаватель С.И.Тимошенко, доц., к.т.н.

**Екатеринбург 2023**

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146061792)

[2 Результаты проделанной работы 6](#_Toc146061793)

[2.1 Задание 1 6](#_Toc146061794)

[2.2 Задание 2 8](#_Toc146061795)

[2.3 Задание 3 12](#_Toc146061796)

[3 Анализ полученных результатов 18](#_Toc146061797)

# 1 Постановка задачи

Задание №1. Используя метрики Холстеда, оценить потенциальное число ошибок в программном обеспечении (ПО) для системы противоракетной обороны (ПРО) типа СОИ (стратегическая оборонная инициатива), энтузиастом которой был бывший президент США Рональд Рейган. Ограничиться расчетами только для одной боевой космической станции (БКС). Необходимые для расчета параметры взять из технического задания (ТЗ) на ПО для БКС:

1. число одновременно сопровождаемых целей равно 20;
2. для поражения каждой цели достаточно сделать 30 измерений каждого отслеживаемого параметра;
3. количество отслеживаемых параметров равно 10;
4. количество рассчитываемых параметров по каждой цели, передаваемых системе уничтожения, равно 3.

Для расчета понадобятся следующие метрики Холстеда:

1. потенциальный объем программы

,

где  – минимальное число различных операндов, в роли которого обычно выступает число независимых входных и выходных параметров;

1. уровень языка программирования  = 1,53 (соответствует языку АДА);
2. потенциальное число ошибок



Задание №2. Используя метрики Холстеда, выполнить для задачи, описанной в задании N 1, расчёты по формулам, описанным ниже:

1. Расчёт структурных параметров:
2. число модулей программного средства: ;
3. если k >> 8, то структура ПО будет многоуровневой (иерархической) с числом уровней в этом случае число модулей рассчитывается как .
4. Расчет длины программы:.
5. Расчет объема программного обеспечения: 
6. Расчет количества команд ассемблера: 
7. Расчет календарного времени программирования: 

где m – количество программистов в бригаде разработчиков; – производительность (число отлаженных команд в день), устанавливаемая директивно (обычно задается в пределах от 10 до 30); m и  задать самостоятельно.

1. Расчет потенциального количества ошибок: 
2. Расчет начальной надёжности ПО (времени наработки на отказ):  где *Тк* для соблюдения размерности следует перевести из количества дней в соответствующее количество часов (исходя из установленной длительности рабочего дня).

В отчете по лабораторной работе в анализе результатов необходимо пояснить причину разных значений для параметра В, полученных при выполнении задания №1 и №2.

Задание №3. Используя метрики Холстеда, оценить текущий рейтинг программиста и число ожидаемых потенциальных ошибок, которое он может допустить в разрабатываемом ПО. Для расчета использовать следующие рекуррентные соотношения:

1. рейтинг в i-й момент времени:

,

где n – количество написанных программистом программ за оцениваемый период;

m – количество программ, в которых найдены ошибки этого программиста;

Vj – объем j-й программы в Кбайт;

Bk – количество допущенных ошибок в к-й программе;

 – коэффициент считать обратно пропорциональным параметрам в скобках;

проверить следующие три варианта коэффициентов:

= 1/(λk+Rk),

= 1/(λk×Rk),

= 1/λk+1/Rk;

1. R0 считать равным 1000;
2. ожидаемое число ошибок равно;
3. считать уровень языка программирования = 1,53 (соответствует языку АДА);
4. за оцениваемый период программистом сделано 4 программы объемом 5, 7, 9 и 11 Кбайт соответственно; в трех предыдущих программах допущено соответственно 2, 5 и 4 ошибки; предполагается написать программу объемом 15 Кбайт.

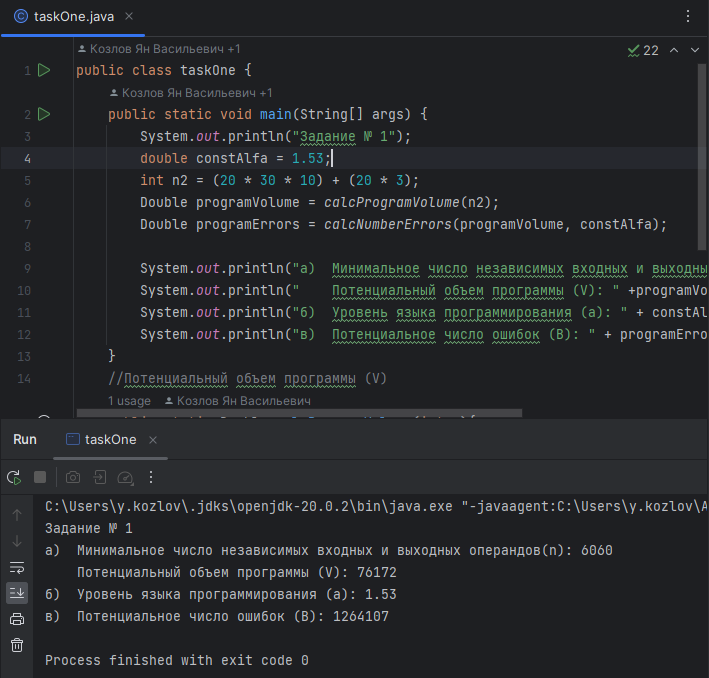
2 Результаты проделанной работы

## 2.1 Задание 1

Согласно поставленной задаче, была разработана программа, представленная ниже.

public class taskOne {  
 public static void main(String[] args) {  
 System.*out*.println("Задание № 1");  
 double constAlfa = 1.53;  
 int n2 = (20 \* 30 \* 10) + (20 \* 3);  
 Double programVolume = *calcProgramVolume*(n2);  
 Double programErrors = *calcNumberErrors*(programVolume, constAlfa);  
  
 System.*out*.println("а) Минимальное число независимых входных и выходных операндов(n): " + n2);  
 System.*out*.println(" Потенциальный объем программы (V): " +programVolume.intValue());  
 System.*out*.println("б) Уровень языка программирования (a): " + constAlfa);  
 System.*out*.println("в) Потенциальное число ошибок (B): " + programErrors.intValue());  
 }  
 //Потенциальный объем программы (V)  
 public static Double calcProgramVolume(int n){  
 return (n + 2) \* (Math.*log*(n + 2) / Math.*log*(2));  
 }  
 //Потенциальное число ошибок (B)  
 public static Double calcNumberErrors(double v, double a){  
 return Math.*pow*(v, 2) / (3000 \* a);  
 }  
}

Результат работы программы представлен на рисунке 1.



1. Результат работы программы 1

Результатом расчета, выполненного программой, является:

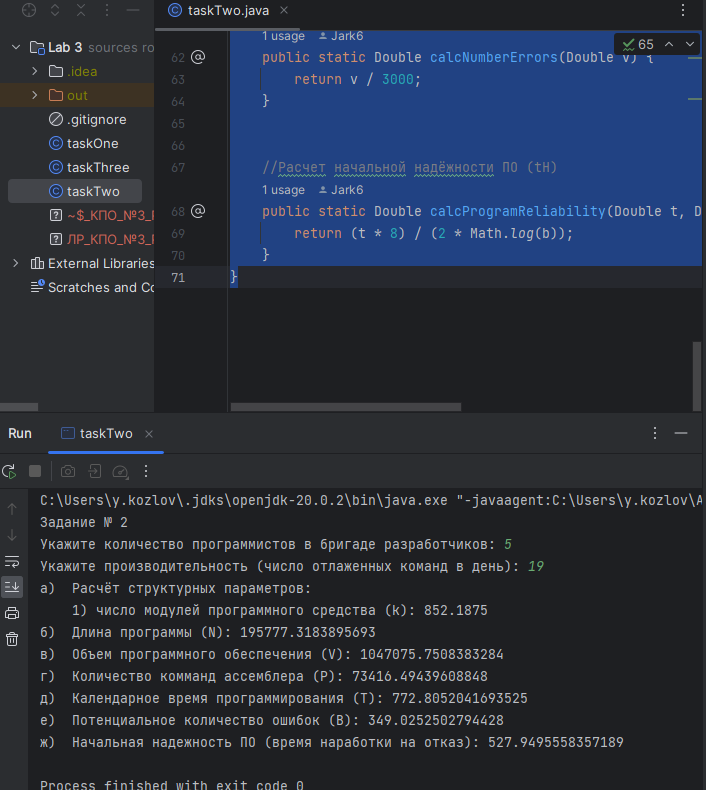
1. потенциальный объем программы (V): 76172;
2. потенциальное число ошибок (B): 1264107.

## 2.2 Задание 2

Согласно поставленной задаче была разработана программа, текст, которой представлен далее.

import java.util.Scanner;  
  
public class taskTwo {  
 public static void main(String[] args) {  
 System.*out*.println("Задание № 2");  
 //double constAlfa = 1.53;  
 int n2 = (20 \* 30 \* 10) + (20 \* 3);  
  
 Scanner in = new Scanner(System.*in*);  
 System.*out*.print("Укажите количество программистов в бригаде разработчиков: ");  
 int programmerAmount = in.nextInt();  
 System.*out*.print("Укажите производительность (число отлаженных команд в день): ");  
 int programmerProductivity = in.nextInt();  
  
 System.*out*.println("а) Расчёт структурных параметров: ");  
 double modulesAmount = *calcModulesAmount*((double) n2);  
 System.*out*.println(" 1) число модулей программного средства (k): " + modulesAmount);  
 double programLength = *calcProgramLength*(modulesAmount);  
 System.*out*.println("б) Длина программы (N): " + programLength);  
 double programVolume = *calcProgramVolume*(modulesAmount);  
 System.*out*.println("в) Объем программного обеспечения (V): " + programVolume);  
 double assemblerCommands = *calcAssemblerCommands*(programLength);  
 System.*out*.println("г) Количество комманд ассемблера (P): " + assemblerCommands);  
 double programmingTime = *calcProgrammingTime*(programLength, programmerAmount, programmerProductivity);  
 System.*out*.println("д) Календарное время программирования (T): " + programmingTime);  
 double programErrors = *calcNumberErrors*(programVolume);  
 System.*out*.println("е) Потенциальное количество ошибок (B): " + programErrors);  
 double programReliability = *calcProgramReliability*(programmingTime, programErrors);  
 System.*out*.println("ж) Начальная надежность ПО (время наработки на отказ): " + programReliability);  
 }  
  
 //расчёт числа модулей программного средства (k)  
 public static Double calcModulesAmount(Double n) {  
 if ((n / 8) > 8)  
 return (n / 8) + (n / (Math.*pow*(8, 2)));  
 else  
 return n / 8;  
 }  
  
 //расчёт длины программы (N)  
 public static Double calcProgramLength(Double k) {  
 return (220 \* k) + k \* (Math.*log*(k) / Math.*log*(2));  
 }  
  
 //Расчет объема программного обеспечения (V)  
 public static Double calcProgramVolume(Double k) {  
 return k \* 220 \* (Math.*log*(48) / Math.*log*(2));  
 }  
  
 //Расчет количества команд ассемблера (P)  
 public static Double calcAssemblerCommands(Double n) {  
 return (3 \* n) / 8;  
 }  
  
 //Расчет календарного времени программирования(T)  
 public static Double calcProgrammingTime(Double n, Integer m, Integer v) {  
 return (3 \* n) / (8 \* m \* v);  
 }  
  
 //Расчет потенциального количества ошибок(B)  
 public static Double calcNumberErrors(Double v) {  
 return v / 3000;  
 }  
  
  
 //Расчет начальной надёжности ПО (tH)  
 public static Double calcProgramReliability(Double t, Double b) {  
 return (t \* 8) / (2 \* Math.*log*(b));  
 }  
}

Результат работы программы представлен на рисунке 2.



1. Результат работы программы 2

Результатом расчета, выполненного программой является:

1. Расчёт структурных параметров ( число модулей программного средства (k)): 852.1875;
2. Длина программы (N): 195777.3183895693
3. Объем программного обеспечения (V): 1047075.7508383284;
4. Количество команд ассемблера (P): 73416.49439608848;
5. Календарное время программирования (T): 772.8052041693525 (в расчете заданы условия: количество программистов в бригаде – 5 (m) и число отлаженных команд в день - 19 (v) );
6. Потенциальное количество ошибок (B): 349.0252502794428;
7. Начальная надежность ПО (время наработки на отказ): 527.9495558357189.

Потенциальное количество ошибок отличается от предыдущего результата, так как в программе 1 учитывался уровень языка программирования, что позволяет оценить потенциальное количество ошибок еще до этапа проектирования.

Также можно заметить, что при использовании многоуровневой структуры ПО и множестве модулей значительно увеличивается объем программы и в то же время многократно уменьшается количество потенциальных ошибок.

## 2.3 Задание 3

Согласно поставленной задаче была разработана программа, текст, которой представлен ниже.

import java.util.ArrayList;

import java.util.List;

public class taskThree {

public static void main(String[] args) {

System.out.println("Задание № 3");

List<Integer> programVolume = new ArrayList<>() {{

add(5);

add(7);

add(9);

add(11);

}};

List<Integer> programErrors = new ArrayList<>() {{

add(0);

add(2);

add(5);

add(4);

}};

double lambda = 1.53;

// Вычисляем для трех вариантов коэффициентов

for (int i = 1; i <= 3; ++i) {

double rating = calcRating(programVolume, programErrors, lambda, i);

System.out.println("Рейтинг для коэффициента " + i + ":");

System.out.printf("%.4f %n", rating);

// Вычисляем ожидаемое число ошибок

int expectedErrors = (int)(getCoef(lambda, rating, i) \* 15);

System.out.println("Ожидаемое число ошибок в следующей программе объемом 15 Кб: ");

System.out.printf("%d шт. %n", expectedErrors);

}

}

// Вычисление варианта коэффициента c(lambda\_k, Rk)

public static double getCoef(double lambda, double Rk, int var) {

switch (var) {

case 1 -> {

return (1.0 / (lambda + Rk));

}

case 2 -> {

return (1.0 / (lambda \* Rk));

}

case 3 -> {

return ((1.0 / lambda) + (1.0 / Rk));

}

default -> {

return 0;

}

}

}

// Рейтинг в i-й момент времени

// i = programVolume.size(), т.к. для n написанных программ можно вычислить рейтинг в n-ый момент времени

// размер programErrors предполагается таким же, т.е. по каждой программе есть данные о количестве ошибок в ней

public static Double calcRating(List<Integer> programVolume, List<Integer> programErrors, double lambda, int var) {

int i = programVolume == null ? 0 : programVolume.size();

// Базовый случай при i = 0: R0 = 1000

if (i == 0) return 1000.0;

// Рекурсивный вызов: вычисляем предыдущий рейтинг

// списки программ и ошибок в них сокращаются на 1 элемент

double prevRate = calcRating(programVolume.subList(0, i - 1), programErrors.subList(0, i - 1), lambda, var);

// Сумма объемов программ

Integer programVolumeSum = programVolume.stream().reduce(0, Integer::sum);

// Сумма ошибок с коэффициентом c

double programErrorSum = 0;

for (int k = 0; k < i; k++) {

double Bk = programErrors.get(k); // количество ошибок

// Получаем коэффициент c, для его вычисления используем вычисленный по рекурсивной формуле рейтинг

// в момент времени k - 1

double c = getCoef(

lambda,

calcRating(programVolume.subList(0, k), programErrors.subList(0, k), lambda, var),

var

);

programErrorSum += Bk / c;

}

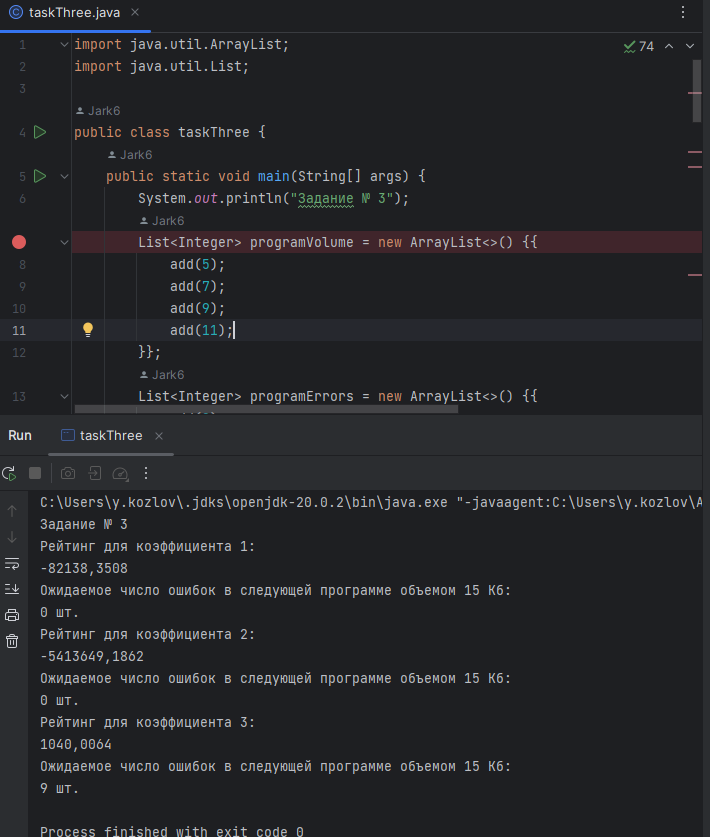
// Окончательно вычисляем рейтинг по формуле

return prevRate \* (1.0 + 1E-3 \* (programVolumeSum - programErrorSum));

}

}

Результат работы программы представлен на рисунке 3.



1. Результат работы программы 3

Программой были рассчитаны 3 рейтинга программиста на основе различных коэффициентов и исходя из этого были даны 3 варианта количества ошибок в программе.

1. Рейтинг для коэффициента 1 составил -82138,3508. Отрицательный результат показывает аномальный результат для данного коэффициента, поэтому ожидаемое число ошибок в данном варианте посчитать не представляется возможным;
2. Рейтинг для коэффициента 2 составил -5413649,1862. В данном случае также продемонстрирован отрицательный результат для коэффициента, что является отклонением. Ожидаемое число ошибок в данном варианте также посчитать не удалось;
3. Рейтинг для коэффициента 3 составил 1040,0064. Для данного рейтинга было посчитано ожидаемое число ошибок в следующей программе объемом 15 Кб и оно составило 9 шт.

# 3 Анализ полученных результатов

В результате работы были изучены материалы и разработано программное обеспечение, выполняющее расчеты при любых заданных входных параметрах для заданий №1 – №3, использующих метрики Холстеда.

На основании анализа результатов различных методик расчета количества потенциальных ошибок в заданиях №1 и №2 сделан вывод, что использование многоуровневой (иерархической) структуры позволяет обеспечить значительное уменьшение потенциального количества ошибок.

В задании №3 приемлемый результат расчета рейтинга и потенциального количества ошибок возможен только для третьего варианта коэффициента.