ПЗ 1. Количественные параметры программ по модели Холстеда («Метрики Холстеда»)

(Составили Никонова Г.В. ngvlad@mail.ru, Никонов А.В. nalva@mail.ru)

ЭЛЕКТРОННЫЙ ВАРИАНТ

https://disk.yandex.ru/d/xgBQ5OhfmR1fdg

Тематика: количественные параметры программ по модели Холстеда, связанные с размером программ.

Задание. 1. Представить графически схему алгоритма программы. Установочный материал см. в приложении А.

2. Определить количественные параметры, связанные с размером программы:

1) число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

2) число уникальных операндов программы (словарь операндов);

3) общее число операторов в программе;

4) общее число операндов в программе;

5) алфавит (**словарь**) программы;

6) экспериментальную длину программы;

7) теоретическую длину программы;

8) объём программы;

9) потенциальный объём программы;

10) уровень реализации программы;

11) сложность программы;

12) ожидаемый уровень реализации программы;

13) интеллект программы;

14) работу по программированию;

15) время кодирования;

16) ожидаемое время кодирования;

17) уровень языка программирования;

18) уровень ошибок в программе;

19) оптимальная модульность реализации алгоритма в программе.

Пример **расчёта параметров реализации алгоритма** в программе приведён **в приложении Б**.

3. Сделать выводы по работе, см. **приложение Б.**

1 ВВОДНЫЙ МАТЕРИАЛ

Метрики (методы и способы определения) **параметров программ, базируются на определении количественных параметров** [1, 2], отличаются относительной простотой. В их число входит **набор метрик Холстеда** [3].

Метрики по модели **ориентированы на анализ исходного текста** программ. Поэтому они **могут использоваться для оценки сложности промежуточных продуктов разработки**.

В тексте программы можно идентифицировать все операнды, определённые как переменные или константы. Аналогичным образом идентифицируются все операторы, определённые как символы или комбинации символов.

Исходя из идентификации операторов и операндов, можно определить ряд измеримых категорий, обязательно присутствующих в версиях любого алгоритма. Они определяются метриками, с помощью которых могут быть получены числовые значения параметров для основных характеристик качества программ.

Основу **метрик Холстеда составляют четыре измеряемых параметра** программы:

n_1 **– число уникальных операторов программы**, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

n_2 – **число уникальных операндов программы** (словарь операндов);

N_1 – **общее число операторов** в программе;

N_2 – **общее число операндов** в программе.

Опираясь на эти характеристики, получаемые непосредственно **при анализе исходных текстов программ**, **Морис Холстед ввёл следующие оценки**:

– **словарь программы** $n = n_1 + n_2$,

– **длину программы** (экспериментальную длину) $N = N_1 + N_2$,

– **объём программы** $V = N \cdot log_2(n)$ [бит].

Под битом подразумевается **логическая единица информации – символ, оператор, операнд**.

И далее **М. Холстед ввёл величину** n^* – **теоретический словарь программы**, т. е. **словарный запас, необходимый для написания программы**, с учётом того, что **необходимая функция уже реализована** в данном языке и, следовательно, **программа сводится к вызову этой функции**.

Кроме того, модель Холстеда также **прогнозирует количество ошибок в программе в зависимости от её объёма**.

Оценка параметров по модели Холстеда выполняется в следующем порядке.

1. Определяется число уникальных операторов программы n_1 – это **символы-разделители, имена процедур и знаки операций (словарь операторов). К словарю операторов относят** следующие элементы языка:

– имена арифметических и логических операций;

– присваивания;

– условные и безусловные переходы;

– разделители;

– скобки (парные);

– имена процедур и функций;

– выражения типа BEGIN...END, IF...THEN...ELSE, DO...WHILE.

Выражения типа **BEGIN...END, IF...THEN...ELSE, DO...WHILE и им подобные**, осуществляющие блочную группировку операторов, **рассматриваются как единые операторы** (**то же относится и к парам скобок**).

Сюда же относятся ключевые слова следующих категорий, и они **интерпретируются как операторы**:

– **идентификаторы класса памяти** языка (*пример*): inline, register, static, typedef, virtual, mutable;

– **квалификаторы типа** языка (ключевые слова) (*пример*): const, friend, volatile;

– **зарезервированные слова** языка (*пример*): asm, break, case, class, continue, default, delete, do, else, enum, for, goto, if, new, operator, private, protected, public, return, sizeof, struct, switch, this, union, while, namespace, using, try, catch, throw, const_cast, static_cast, dynamic_cast, reinterpret_cast, typeid, template, explicit, true, false, typename;

– **операторы языка программирования** (*пример как иллюстрация*): **! != % %= &&& || &= ( ) * *= + ++ += , - -- -= → . … / /= : :: <<<<= <= = == >>= >>>>= ? [ ] ^ ^= { } | |= ~** и т. д.;

– **управляющие структуры** языка, которые **интерпретируются как один оператор**: это **for(…), if (…), switch (…), while for (…) and catch (...)**. *Например*,

`for (var col = 0; col < 4; col++)` – **это один оператор**.

2. Определяется число уникальных операндов программы n_2 – это **словарь операндов**. При расчёте метрик Холстеда используются следующие **операнды:**

– **идентификаторы** – все **идентификаторы, которые не являются зарезервированными словами**;

– **идентификаторы типов** – это зарезервированные **слова, обозначающие тип данных** языка (*пример как иллюстрация*): **`bool`**, **`char`**, **`double`**, **`float`**, **`int`**, **`long`**, **`short`**, **`signed`**, **`unsigned`**, **`void`**;

– **константы** в программе (*пример как иллюстрация*) – **числовые, символьные и строковые**.

Например (язык **C++**)**:**

```
a = b + c; // a, b, c - это операнды; =, + - это операторы;
a = (b>>2); /* 2 здесь является константой, и необходимо решать, может ли она считаться как операнд алгоритма, участвуя в операции сдвига;
   >> - это оператор - побитовый сдвиг вправо. */
```

3. Определяется общее число операторов N_1 в программе.

4. Определяется общее число операндов N_2 в программе.

Данные для пп. 1–4 получают непосредственно при анализе исходных текстов программ (листингов).

Также М. Холстедом введены следующие оценки, которые нужно определить.

5. Определяется алфавит (словарь) программы: $n = n_1 + n_2$.

6. Определяется экспериментальная длина программы: $N_Э = N_1 + N_2$.

Для стилистически корректных программ, по Холстеду, **отклонение $N_Э$ от теоретической длины** программы не превышает **10 %**.

7. Определяется теоретическая длина программы: $N_T = n_1 log_2 n_1 + n_2 log_2 n_2$.

Метрики теоретической длины и экспериментальной длины программы можно использовать **для выявления недостатков** программирования. Если теоретическое значение длины программы и экспериментальное значение длины программы **отличаются более чем на 10 %**, то **возможно наличие** в программе отдельных **несовершенств**.

8. Определяется объём программы (объём кода запрограммированного алгоритма): $V = N_Э \cdot log_2(n)$ **[бит]**.

Обычно, **объём программы по метрике Холстеда может укладываться** в следующие диапазоны [3]:

до 10: очень небольшие программы;

10–100: небольшие программы;

100–1000: средние программы;

1000–10000: крупные программы;

более 10000: очень крупные программы.

Размер найденной меры (числового значения) **для одной функции рекомендуется от 20 до 1000, а для одного файла от 100 до 8000**. Если размер метрику превышает верхнюю границу диапазона, то рекомендуется пересмотреть функциональную нагрузку на исследуемый элемент и разбить его на несколько составляющих, или провести оптимизацию запрограммированного алгоритма.

9. Определяется потенциальный объём программы (соответствующий максимально компактному тексту программы): $V^* = (\eta_2^* + 2) \cdot log_2(\eta_2^* + 2)$,

где η_2^* **– количество имён входных и выходных переменных, представленных в предельно краткой записи** (с точки зрения алгоритмической

сложности – сжатой). Здесь: «**входные переменные**» – то, **что передаётся программе снаружи или задаётся как исходные данные**. «**Выходные переменные**» – это то, **что программа вычисляет и выводит.**

Потенциальный объём программы – **основной исходный параметр**, на котором базируются все расчёты метрических характеристик.

Например, для задания **одномерного массива** (т. е. **строки**), каково бы ни было число его элементов, **требуется всего два имени**:

– **указатель адреса начала массива**;

– **количество элементов в нем**.

Точно так же для задания **двумерного массива** достаточно иметь **три параметра**:

– **указатель адреса первого элемента**;

– **число столбцов**;

– **число строк**.

Эта **величина называется потенциальным объёмом** (**минимально возможным**), **соответствующим максимально компактному тексту программы**. Это объясняется тем, что **в потенциальном языке минимизировано число операторов, а все операнды сведены к перечню процедур или функций и списку входных и выходных переменных**.

10. Определяется уровень реализации (**изготовления**) **программы**: $L = V^*/V$.

Это метрический показатель, который **характеризует степень компактности программы, экономичность использования средств алгоритмического языка**. Чем ближе значение L к единице, тем более совершенна программа.

11. Определяется сложность программы: $S = 1/L$. При переводе алгоритма с одного языка на другой его потенциальный объем V^* не изменяется, но дей-

ствительный объём V может увеличиваться или уменьшаться в зависимости от развитости языков программирования.

Для потенциального языка справедливо равенство $V = V^*$, а для любого менее развитого языка следует учитывать соотношение $V > V^*$. Это обусловлено тем, что для потенциального языка **длина программы и словарь программы соотносятся**: $N^* = n^*$, а для всех других языков применяется уравнение длины и учёт соотношения $N_Э > n$.

12. Определяется ожидаемый уровень (изготовления) программы: $L^{\wedge} = \frac{2}{n_1} \cdot \frac{n_2}{N_2}$.

Это **уровень качества программирования, основанный лишь на параметрах реальной программы без учёта теоретических параметров**.

13. Определяется интеллект программы: $I = L^{\wedge} \cdot V$. Это оценка **необходимых интеллектуальных усилий при разработке** программы, характеризующая «**число элементарных решений**», требуемых при написании программы.

14. Определяется работа по программированию: $E = V \cdot S$.

15. Определяется время кодирования: $T = E/Str$. Здесь обычно $Str = 10$ – **число Страуда** («**число умственных операций в единицу времени**»).

16. Определяется ожидаемое время кодирования (реализации алгоритма):

$$T^{\wedge} = \frac{n_1 N_2 (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 n}{2 n_2 Str},$$

где ***Str*** – **число Страуда**, лежит в интервале от **5-ти до 20**.

Число Страуда определяет **скорость обработки информации человеческим мозгом (Холстед принял эту величину равной 18 – «число умственных операций в единицу времени»).**

Эта теоретическая выкладка М. Холстеда **плохо сочеталась с практикой**, и поэтому данному параметру добавили определение «**ожидаемое**».

17. Определяется уровень языка программирования: $\lambda = L \cdot V^*$.

Холстед установил, что **для любого алгоритма, который программируется на разных языках**, с **увеличением объёма уменьшается уровень реализации в той же пропорции**. В результате **произведение уровня *L* на объём *V* равняется потенциальному объёму *V**** реализуемого алгоритма.

18. Определяется уровень ошибок: $B = V/3000$.

Здесь речь идёт о **«переданных ошибках»**, то есть **ошибках, остающихся в реализации программы после** некоторой **идентифицирующей фазы** (проверки листинга), **такой как публикация** или **переход к следующему этапу разработки** (передача модуля для сборки программного продукта).

19. Определить оптимальную модульность реализации алгоритма программой: $M = \eta_2^*/6$.

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

ПРИЛОЖЕНИЕ А

Графическое представление схемы алгоритма программы

1) представить рисунком схему алгоритма программы, выполненной на знакомом языке программирования (одном языке).

Размер листинга не более одной–двух страниц формата А4, индивидуален для каждого студента.

Пример программы алгоритма схемы приведён на рисунке А.1, [4].

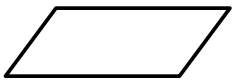
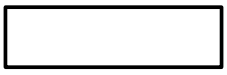
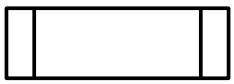
Схема алгоритма программы должна отражать последовательность операций в программе, а не показывать схему работы какой-то системы.

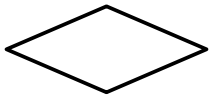
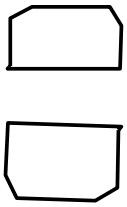
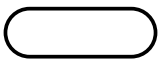
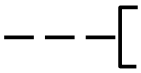
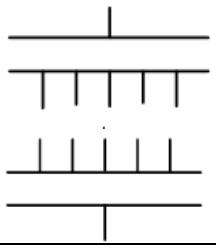
Не использовать синтаксис языка программирования, и при необходимости руководствоваться п. 4.1.4 ГОСТ [4] («Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ»).

Также, п. 4.1.5 ГОСТ [4]: («В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом»).

Условные графические обозначения (УГО) и правила выполнения по ГОСТ – краткая справка в таблице А.1.

Таблица А.1 – УГО в схемах алгоритмов, программ, данных и систем

УГО	Наименование	Назначение
	**Данные**	Определяет ввод или вывод на внешнее устройство или носитель данных.
	**Процесс**	Отражает обработку данных (выполнение операции, группы операций).
	**Типовой процесс**	Отображает предопределенный процесс, состоящий из одной или нескольких операций программы, которые определены в другом месте.
	**Подготовка**	Отображает модификацию параметра для управления циклом со счетчиком.

	**Решение**	Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два или более альтернативных выходов, один из которых активизируется после вычисления условия.
	**Граница цикла**	Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла.
	**Соединитель**	Используется для обрыва линии и продолжения ее в другом месте.
	**Знак завершения**	Определяет начало и конец структурной схемы алгоритма.
	**Комментарий**	Используется для добавления пояснительных записей.
	Основная линия	Отражает последовательность выполнения действий в алгоритме.
	начало **Параллельные действия** конец	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами.

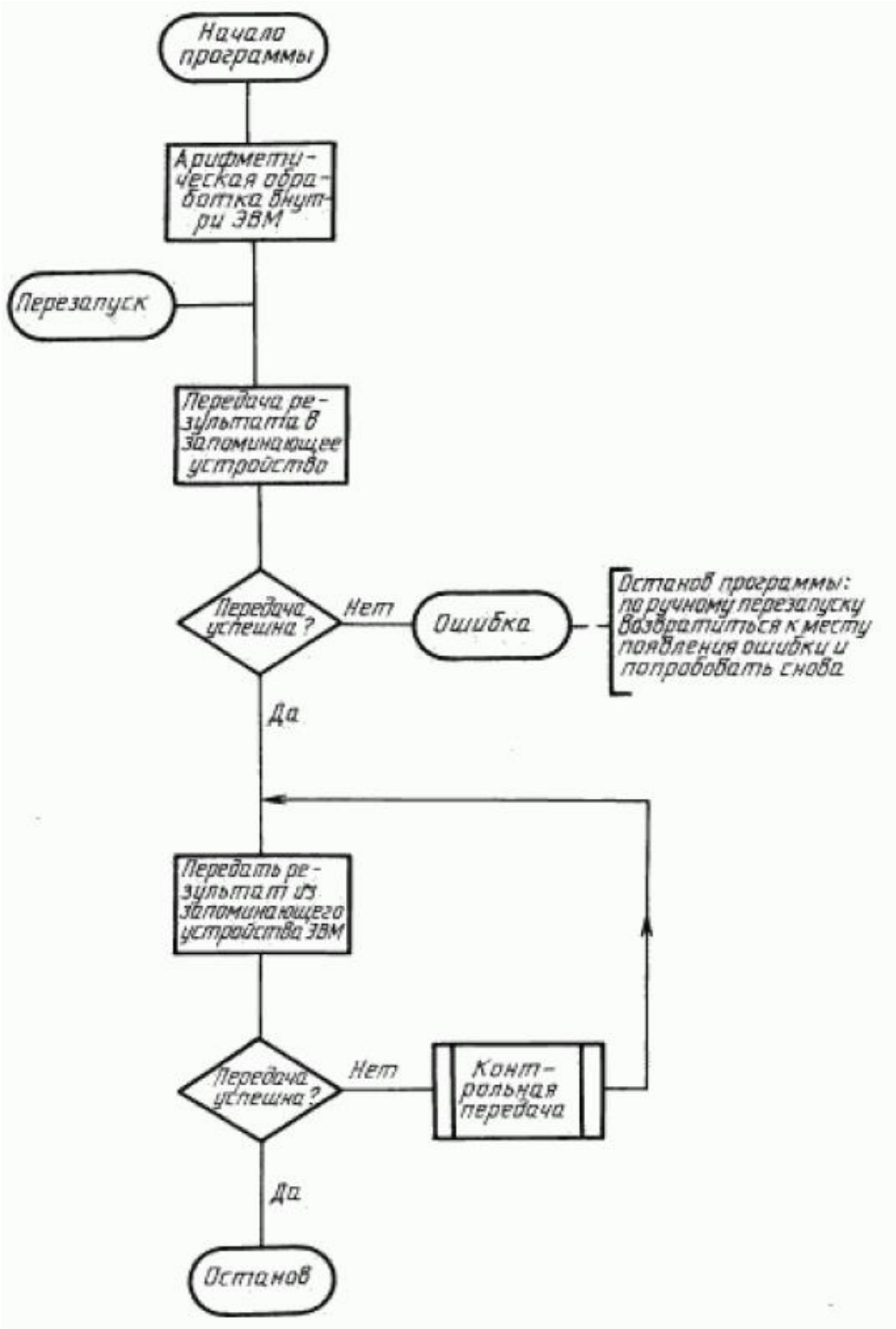


Рисунок А.1 – Пример схемы программы (алгоритма)

ПРИЛОЖЕНИЕ Б

Расчёт параметров реализации алгоритма в программе

Проанализировать листинг, опираясь на синтаксис языка (*пример* операторов – таблица Б.1). Листинг, рассмотренный в данном примере, показан после таблицы Б.1.

Листинг приводить в приложении к основной части отчёта по ПЗ1, а не в теле его основной части.

Пример **анализа листинга приведён в таблицах Б.2 и Б.3.**

Таблица Б.1 – Пример операторов для анализа листинга (нужно знать, а в отчёте не приводить)

Оператор	Операция
!	(НЕ)
!=	**(**Не равно)
<=>	(«–1» если левая часть меньше правой, 0 если равна, 1 если больше)
%	(Остаток от деления. Взятие по модулю (остаток)
%=	(Остаток от деления, совмещённый с присваиванием)
=!	(Был оператором «не равно», теперь означает «назначить логический обратный», как в a = !b)
&&	(Если левое выражение возвращает false, правое не выполняется)
&&&	(В информационно-поисковых системах применяется для расширения границ поиска: поиск будет производиться не в пределах одного документа, а в пределах одного сайта. В «C» нет оператора или символа &&&. Но существуют операторы && (логический «и») и & (унарный адрес объекта или бит «и»).)
|	(«Побитовое ИЛИ»)
||	(ИЛИ – если первый операнд имеет значение (true), то выражение сразу становится (true) и второй операнд даже не рассматривается. В информационно-поисковых системах сложному запросу, состоящему из двух запросов, соединенных этим оператором, удовлетворяют все документы, удовлетворяющие хотя бы одному из этих двух запросов.)
|=	(Присваивает левому операнду результат работы «Побитового ИЛИ»)
&=	(Побитовое «И», совмещённое с присваиванием)
( )	(В «C#» есть способ приведения типов: это оператор «(тип)». Например: object obj = "Некий объект"; string str = (string)obj;)
*****	(Умножение)
****	(Возведение в степень в Perl)
***=**	(Умножение, совмещённое с присваиванием)

+	(Сложение)
++	(Инкремент – операция, увеличивающая переменную)
−−	(Декремент – операция, уменьшающая переменную)
+=	(Сложение, совмещённое с присваиванием)
,	(Оператор «запятая». В скалярном контексте выполняется левый аргумент, результат игнорируется, затем правый и его результат есть результат действия оператора. В списковом контексте это разделитель элементов списка, который включает указанные элементы в список.)
−	(Вычитание, а также унарный минус)
−=	(Вычитание, совмещённое с присваиванием)
→	(Арифметический сдвиг вправо. Операция доступа к элементам: доступ осуществляется через указатель на объект, что чаще всего практикуется для доступа к компонентам в C++ Builder.)
.	(Операция доступа к элементам: доступ осуществляется через объект.)
..	(Оператор диапазона)
...	(Для некоторых функций невозможно указать количество и типы всех аргументов. Список аргументов в объявлениях таких функций заканчивается многоточием, что означает «и может быть ещё несколько аргументов».)
/	(Деление)
/=	(Деление, совмещённое с присваиванием)
\	(Получение ссылки на переменную)
;	(Пустой оператор)
:	(Указывает тип переменной. В операторах выбора из нескольких альтернатив отделяет описание условий применимости отдельного случая от действий, которые при этом должны выполняться. Может использоваться для того, чтобы явным образом формировать векторы. Чтобы выделить выбранные строки, столбцы и элементы векторов и матриц. ...)
::	(Оператор разрешения области: используется для идентификации и устранения неоднозначности идентификаторов, которые используются в разных областях.)
>>>	(Оператор сдвига вправо без знака у Java. Освобождающиеся разряды заполняются нулями.)
<<	(Сдвиг влево побитно)
>>=	(Присваивает левому операнду результат работы оператора «Сдвига вправо»)
<<<<=	(Присваивает правому операнду результат работы оператора «Сдвига влево». В Java потребовался беззнаковый сдвиг, потому что в C и C++ сдвиг всегда беззнаковый. То есть <<<< в Java – это сдвиг вправо в C и C++. Но, поскольку в Java все численные типы со знаком (за исключением `char`), то и результаты сдвигов должны иметь знаки.)
<=	(Арифметическое «меньше или равно»)
=	(Оператор «присвоить» правое значение переменной слева)
==	(Результат true если левая часть равна правой (равно)

?	(Оператор работает как и в «С». Если выражение перед «?» истинно, то выполняется аргумент перед «:» - иначе после «:».)
[ ]	(Доступ к элементам массива или контейнера. Тип возвращаемого значения обычно является ссылкой на то, что хранится в контейнере.)
^	(Побитное XOR («Исключающее ИЛИ»)
^=	(Оператор присваивания побитового исключающего «ИЛИ» («XOR»)
{ }	(Это «Составной оператор – {...}». Операторы могут быть сгруппированы в «группу-оператор» путём инкапсуляции группы операторов с фигурными скобками. Его действие состоит в последовательном выполнении содержащихся в нем операторов, за исключением тех случаев, когда какой-либо оператор явно передает управление в другое место программы.)
~	(«Побитовое НЕ»)
	и так далее...

Ниже **представлен листинг программного модуля, используемого в данном** *примере*. (**В отчёте привести отдельным приложением в конце текста отчёта**). Программа написана **на языке C#**.

```
usingSystem;
 usingSystem.Collections.Generic;
 usingSystem.Linq;
 usingSystem.Text;
 usingSystem.Threading.Tasks;
 namespace ConsoleApplication1
 {
 class Program
    {
 static void Main(string[] args)
 {
          //Ввод массива
 Console.WriteLine("Введитеразмерность");
 int M = Convert.ToInt32(Console.ReadLine());
 int N = Convert.ToInt32(Console.ReadLine());

 //Заполнение массива сл. числом и вывод массива
 Random rand = new Random();
 int[,] mass = new int[M, N];

 for (inti = 0; i< M; i++)
          {
 for (int j = 0; j < N; j++)
             {
 mass[i, j] = rand.Next(0, 10);
 Console.Write(mass[i, j]);
 }
 Console.WriteLine();
          }
```

```
            //Проход массива по строкам
intpovtorenie = 0;
int[] povtoreniaya_v_stroke = new int[M];

for (inti = 0; i< M; i++)
            {
for (int j = 0; j < N - 1; j++)
                {
for (int k = j + 1; k < N; k++)
if (mass[i, j] == mass[i, k])
povtorenie++;
                }
Console.WriteLine("выводповторения в " + i + " строке = " + povtorenie);
povtoreniaya_v_stroke[i] = povtorenie;
povtorenie = 0;
            }

            //Вывод номера строки, где povtorenie больше всего
intnomer = 0;

for (inti = 0; i< M; i++)
            {
for (int j = i + 1; j < M - 1; j++)
                {
intmaxValue = povtoreniaya_v_stroke.Max();
if (povtoreniaya_v_stroke[i] == maxValue)
                    {
nomer = i;
                    }
                }
            }
Console.WriteLine(" Строка, гдебольшевсегоповторений " + nomer);
Console.ReadKey();
        }
    }
}
```

Экспериментально определённые исходные для расчётов данные представить в виде таблиц (*например*, таблицы Б.2 и Б.3).

Таблица Б.2 – Таблица операторов программы (*пример*) **!привести в отчёте!**

Оператор	Уникальный n_1	N_1
;	1	33
{}	2	10
()	3	23
=	4	19
==	5	2

+	6	6
++	7	8
-	8	2
[]	9	9
for (inti = 0; i<M; i++)	10	3
for (int j = 0; j < N; j++)	11	1
for (int j = 0; j < N - 1; j++)	12	1
for (int k = j + 1; k < N; k++)	13	1
if (mass[i, j] == mass[i, k])	14	1
for (int j = i + 1; j < M - 1; j++)	15	1
if (povtoreniaya_v_stroke[i] == maxValue)	16	1
int	17	13
Random	18	2
new	19	3
Console.WriteLine	20	4
Convert.ToInt32	21	2
Console.ReadLine	22	2
Console.Write	23	1
Console.ReadKey	24	1
<	25	7
void	26	1
static	27	1
	27	155
	n_1	N_1

Таблица Б.3 – Таблица операндов программы (*пример*) **!привести в отчёте!**

Операнд	Уникальный n_2	N_2
string	1	1
M	2	7
N	3	5
i	4	17
j	5	10
k	6	2

rand	7	2
mass	8	5
povtorenie	9	4
povtoreniaya_v_stroke	10	3
nomer	11	3
0	12	9
10	13	1
1	14	4
maxValue	15	2
" Строка, где больше всего повторений "	16	1
"выводповторения в "	17	1
" строке = "	18	2
	18	79
	n_2	N_2

На основе экспериментальных данных из составленных вами таблиц Б.2 и Б.3 рассчитать параметры программы, и результаты расчёта представить в таблице Б.4 (*пример*).

Таблица Б.4 – Пример исходных данных для определения значений мер по Холстеда !привести в отчёте!

Название данных	Частота данных
Число уникальных операторов (n_1)	27
Число уникальных операндов (n_2)	18
Общее число операторов (N_1)	155
Общее число операндов (N_2)	79
Словарь программы (n)	45
Кол-во вход., вых. переем (η_2^*)	3

К переменным η_2^* отнесены следующие переменные: .....

Рассчитать параметры программы по модели Холстеда. *Пример* **расчёта значений мер Холстеда по приведённому выше листингу, показан ниже.**

В выводах предложить смысловую трактовку как используемых данных, так и описываемых характеристик.

Пример **расчёта метрик Холстеда по приведённому выше листингу, следующий.**

Учитывая исходные данные, таблица Б.4, получены следующие оценки характеристик программы по модели Холстеда:

– словарь программы:

$$n = n_1 + n_2 = 27 + 18 = 45; \quad (1)$$

– длина программы:

$$N = N_1 + N_2 = 155 + 79 = 234; \quad (2)$$

– объём программы:

$$V = N\log_2(n) = 234 \cdot \log_2(45) = 1285{,}09 \text{ бит}; \quad (3)$$

– теоретическая длина программы:

$$N_T = n_1\log_2(n_1) + n_2\log_2(n_2) = 27 \cdot \log_2(27) + 18 \cdot \log_2(18) = 203{,}44; \quad (4)$$

– экспериментальная длина программы:

$$N_Э = N_1 + N_2 = 155 + 79 = 234; \quad (5)$$

– потенциальный объем программы:

$$V^* = (n_2^* + 2)\log_2(n_2^* + 2) = (3 + 2)\log_2(3 + 2) = 11{,}60 \text{ бит}; \quad (6)$$

– уровень качества программирования (реализации программы):

$$L = V^*/V = 11{,}6096/1285{,}09 = 0{,}009; \quad (7)$$

– ожидаемый уровень реализации программы (основанный лишь на параметрах реальной программы без учёта теоретических параметров):

$$L\hat{} = (2n_2)/(n_1N_2) = (2 \cdot 18)/(18 \cdot 79) = 0{,}0169; \quad (8)$$

– интеллект программы:

$$I = L\hat{}V = 0{,}0169 \cdot 1285{,}09 = 21{,}69; \quad (9)$$

– работа по программированию:

$$E = VS = 1285{,}09 \cdot 111{,}11 = 142249{,}5 \text{ [количество элементарных операций по осмыслению]}, \quad (10)$$

где $S = 1/L = 1/0{,}009 = 110{,}69$;

– время кодирования:

$$T = E/Str = 142249{,}5/10 = 14224{,}95 \text{ с}, \quad (11)$$

где Str = 10 – число Страуда, т. е. число умственных операций в единицу времени;

– ожидаемое время кодирования:

$$T^{\wedge} = \frac{n_1 N_2 (n_1 \log_2 n_1 + n_2 \log_2 n_2) \log_2 n}{2 n_2 Str} =$$

$$= \frac{27 \times 79 (27 \log_2 27 + 18 \log_2 18) \log_2 45}{2 \cdot 18 \cdot 10} = 7614{,}18 \text{ с}; \quad (12)$$

– уровень языка программирования:

$$\lambda = LV^{*} = 0{,}009 \cdot 11{,}6096 = 0{,}0008; \quad (13)$$

– ожидаемый уровень ошибок:

$$B = V/3000 = 1285{,}09/3000 = 0{,}4284, \quad (14)$$

где 3000 – среднее число элементарных различений (бит информации) между возможными ошибками при программировании;

– оптимальная модульность:

$$M = \eta_2^{*}/6 = 3/6 = 0{,}5; \quad (15)$$

Для наглядности результаты представлены в таблице 6.

Таблица 6 – Рассчитанные значения метрик по модели Холстеда **!Указать все параметры согласно задания!**

Название метрики	Результат
Теоретическая длина программы $N_т$	203,44
Экспериментальна длина программы $N_э$	234
Потенциальный объём V*, бит	11,6096
Объём программы V, бит	1285,09
Уровень качества программирования L	0,009
Уровень программы по параметрам реальной программы L^	0,0169
Интеллект программы I	21,69
Работа по программированию E, кол-во элем-х опе-й по различению	142249,5
Время кодирования T, с	14224,95
Ожидаемое время кодирования T^, с	7614,18
Уровень языка программирования λ	0,0008
Уровень ошибок B	0,4284
Оптимальная модульность M	0,5

Можно сделать следующие выводы.

1. Метрики теоретической длины и экспериментальной длины программы можно использовать для выявления недостатков программирования. Для корректно написанного кода

относительная разность теоретической и экспериментальной длины программы (по Холстеду) лежит в пределах 10–12 %.

В данной программе теоретическое значение длины программы N_T = 203,44 и экспериментальное значение длины $N_Э$ = 234 отличаются более чем на 10 %, и по Холстеду можно утверждать о наличии стилистических ошибок (несовершенств).

2. Размер объёма программы для одной функции рекомендуется от 20 до 1000, а для одного файла – от 100 до 8000. Если размер этой метрики превышает верхнюю границу диапазона, то рекомендуется разбить исследуемый элемент на несколько составляющих.

В данной программе объем составляет 1285,09 и укладывается в рекомендуемый диапазон.

3. Потенциальный объём зависит от количества входных и выходных параметров n_2* = 3 и не зависит от языка реализации. V* = 11 соответствует минимально компактному тексту.

4. Уровень качества программирования – это численное значение, показывающее краткость программы. Чем ближе значение L к единице, тем более совершенна программа. Для идеальной программы L = 1, а для реальной L < 1.

Уровень рассматриваемой программы равен 0,009, что значительно меньше, чем 1. Это свидетельствует о том, что программа является некомпактной. Но метрики Холстеда не были предназначены для объектно-ориентированного программирования.

Но если данную программу рассматривать как функцию «вывода номера первой или последней строки – столбца матрицы», то она некомпактна.

5. Уровень программы по параметрам реальной программы – это уровень качества программирования, основанный лишь на параметрах реальной программы без учета теоретических параметров. Эта метрика применяется, когда нужно определить уровень программирования, не прибегая к оценке её теоретического объема.

Уровень программы по параметрам реальной программы составляет 0,0169. В соотношении с L этот уровень примерно в два раза выше.

6. Интеллект программы – это оценка необходимых интеллектуальных усилий при разработке программы, и метрика характеризует число требуемых элементарных решений при написании программы (оценка мыслительных затрат).

В рассмотренной программе I = 21,69. Это свидетельствует о том, что для создания программы не нужно прикладывать больших умственных затрат.

7. Работа по программированию составляет 142249,5 мыслительных действий (различений). Эта характеристика показывает количество элементарных операций, которые совершает человеческий мозг при реализации программы.

Много это или мало? – судить по временным затратам.

8. Время кодирования T и ожидаемое время кодирования T^ различаются в данном примере практически в два раза: T = 14224,95 ≈ 4 ч, T^ = 7614,18 ≈ 2,1 ч. Это может зависеть от языка программирования. Когда Холстед разрабатывал данные метрики, языки программирования требовали более объемного и сложного алгоритма в реализации, дополнительных переменных и операций.

9. Уровень языка программирования λ = 0,0008. Так как произведение L на V^* остается неизменным для любого языка, следовательно, λ уменьшается пропорционально увеличению объема программы.

10. Ожидаемый уровень ошибок B составляет 0,4284. Данная метрика показывает возможные ошибки, оставшиеся в реализации программы после некоторой идентифицирующей фазы. Поскольку данный метод предсказывает лишь число первоначальных ошибок, есть расхождение при их исключении, и метод не дает «доказательства правильности» в математическом смысле, но обеспечивает высокую степень доверия в прикладном отношении. В примере вероятно не передана и одна ошибка.

Время, требуемое на реализацию программы для ЭВМ, зависит от числа элементарных мысленных различений, необходимых программисту для поиска ошибок. Следовательно, число моментов, в которые можно сделать ошибочное различение при тестировании, также определяется величиной работы при выполнении процесса тестирования.

11. Оптимальная модульность M = 0,5, и она показывает количество модулей, на которые желательно разбить программу. Это принцип, согласно которому логически связанные между собой подпрограммы, переменные и т. д. группируются в отдельные модули. Разбиение на модули упрощает понимание и снижает стоимость разработки. Модульность снижает работу по программированию. Пример не превышает объём одного модуля.

Полученные результаты показывают, что данный программный модуль имеет большое время программирования и большое значение работы по программированию, но в то же время небольшой интеллект программы, низкий уровень качества программирования, низкий уровень языка программирования. Большинство расчётов опирается на потенциальный объем программы.

Метрики Холстеда, как правило, вычисляются уже на основе готового исходного кода и применимы для получения апостериорных оценок, т. е. на основании опыта. Они могут быть использованы для того, чтобы предотвратить чрезмерное усложнение отдельных структурных элементов программного проекта и роста связанных с этим рисков. С помощью этих метрик могут быть получены основные характеристики качества программ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Богданов Д.В., Путилов В.А., Фильчаков В.В. Стандартизация процессов обеспечения качества программного обеспечения / Учебное пособие Апатиты . – Апатиты, КФ ПетрГУ, 1997. – 161 С. : [Электронный ресурс]. – http://abc.vvsu.ru/Books/ebooks_iskt/Электронныеучебники/Метрология_качество_прогр_обеспечения/Качество/index1_4.html.

2. Черников Б.В., Поклонов Б.Е. Оценка качества программного обеспечения. Практикум. – М., ИД «Форум»–ИНФРА–М. – 2012. – 400 с.

3. Холстед М. М. (1977). Практическая теория программирования. Нью-Хейвен: Йельский университет.

4. ГОСТ 19.701− 90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 1990. – 8 с.

5.