Лабораторная работа № 13
«Оценка характеристик программ на основе лексического анализа»

Краткая теория и методические указания

Одной из тем в программировании, к которым интерес периодически то появляется, то пропадает, является вопрос метрик кода программного обеспечения. В крупных программных средах время от времени появляются механизмы подсчета различных метрик. Волнообразный интерес к теме так выглядит потому, что до сих пор в метриках не придумано главного — что с ними делать. То есть даже если какой-то инструмент позволяет хорошо подсчитать некоторые метрики, то что с этим делать дальше зачастую непонятно. Конечно, метрики — это и контроль качества кода (не пишем большие и сложные функции), и «производительность» (в кавычках) программистов, и скорость развития проекта.

Для измерения характеристик программных средств (например, оценки трудоемкости тестирования или сопровождения) используют метрики. В исследовании метрик ПО существует два основных направления:

• поиск метрик, характеризующих специфические свойства программ, т. е. метрик оценки самого программного обеспечения;

• использование метрик для оценки технических характеристик и факторов разработки программ, т. е. метрик оценки условий разработки программ.

В настоящее время в мировой практике используется несколько сотен метрик программ. Далеко не все из них носят прикладной характер, однако многие из них полезны для теоретических и практических оценок трудоемкости разработки и тестирования ПО.

Основными направлениями исследований, для которых применяются метрики, являются:

1) оценки топологической и информационной сложности программ;
2) оценки надежности программных систем;
3) оценки производительности ПО;
4) оценки уровня языковых средств и их применения;
5) оценки возможности понимания программных текстов, что необходимо для сопровождения и модификации программ;
6) оценки производительности труда программистов для составления планов и графиков работ по созданию ПО.

Для тестирования ПО с точки зрения функциональности наиболее важными характеристиками являются характеристики первой группы, поскольку именно от этих характеристик напрямую зависит объем тестирования. Кроме того, эти метрики в большинстве своем достаточно просты, но вполне пригодны для получения хороших результатов.

Имеется достаточно большое количество метрик, позволяющих оценить сложность программного обеспечения. Как правило, метрики сложности делят на три основные группы:

1) метрики размера ПО;
2) метрики сложности потока управления ПО;
3) метрики сложности потока данных ПО.

В общем случае применение метрик позволяет руководителям проектов и предприятий изучить сложность разработанного или даже разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения. Однако метрики могут служить лишь рекомендательными характеристиками, ими нельзя полностью руководствоваться, так как при разработке ПО программисты, стремясь минимизировать или максимизировать ту или иную меру для своей программы, могут прибегать к хитростям вплоть до снижения эффективности работы программы. Кроме того, если, к примеру, программист написал малое количество строк кода или внес небольшое число структурных изменений, это вовсе не значит, что он ничего не делал, а может означать, что дефект программы было очень сложно отыскать. Последняя проблема, однако, частично может быть решена при использовании метрик сложности, т.к. в более сложной программе ошибку найти сложнее.

Метрики сложности потока управления программами

С помощью этих показателей качества в процессе оценок ПО оперируют либо плотностью управляющих переходов внутри программ, либо взаимосвязями этих переходов. В настоящее время в мировой практике используется несколько сотен метрик программ. Рассмотрим основные.

1. Метрики Джилба

Достаточно практичными являются метрики программного обеспечения, предложенные Томасом Джилбом (Thomas Jilb) и также основанные на результатах анализа текстов программных продуктов.

В качестве меры логической трудности Джилб предложил число логических «двоичных принятий решений». Наиболее ценным для практики является то, что такая оценка может быть получена вручную на основе зрительного анализа текста программы, либо автоматически с помощью специально разработанных программных анализаторов, причем относительно несложных.

Абсолютная логическая сложность, по мнению автора метрик, должна задаваться числом необычных выходов из операторов, в которых происходит принятие решений. Джилб предположил, что логическая сложность должна являться значимым, если не определяющим, фактором для оценки стоимости программы на начальных этапах ее проектирования.

Логическая сложность программы Джилб определяет как насыщенность программы условными операторами типа IF-THEN-ELSE и операторами цикла (при этом следует учитывать, что фактическая запись условий и циклов в разных языках программирования может быть представлена в разной форме при сохранении указанного смысла операторов). При этом вводятся следующие характеристики программного средства:

CL - абсолютная сложность программы, характеризуемая количеством операторов условий;

cl - относительная сложность программы, определяющая насыщенность программы операторами условия (т. е. относительная сложность программы *cl* вычисляется как отношение абсолютной сложности *CL* к общему числу операторов *L).*

Кроме указанных показателей, Джилб предложил применять следующие характеристики программы:

- количество операторов цикла L_{loop};
- количество операторов условия L_{IF};
- число модулей или подсистем L_{mod};
- отношение числа связей между модулями к числу модулей

$$f = \frac{N_{SV}^4}{L_{\text{mod}}} \quad (1)$$

- отношение числа ненормальных выходов из множества операторов к общему числу операторов

$$f = \frac{N_{SV}^*}{L} \quad (2)$$

Среди всех показателей качества программ Джилб указывает надежность программы, которую он характеризует как возможность того, что данная программа проработает определенный период времени без логических сбоев. В качестве практической оценки программной надежности автор метрик предлагает рассчитывать как единицу минус отношение числа логических сбоев к общему числу запусков.

Отношение количества правильных данных ко всей совокупности данных приводится Джилбом в качестве меры точности (свободы от ошибок), поскольку он считает, что точность нужна как средство обеспечения надежности программы. Прецизионность определяется как мера того, насколько часто появляются ошибки, вызванные одинаковыми причинами. Джилб оценивает этот показатель дробью, в числителе которой указывается число фактических

ошибок на входе, а в знаменателе - общее число наблюдаемых ошибок, причинами которых явились ошибки на входе. Так, например, если одна ошибка вызывает в течение определенного периода времени появление 50 сообщений об ошибках, то прецизионность равна 0,02.

1.2.Пример: Задача «Копирование элементов массива»

1.2.1 Задание

Требуется разработать функцию, которая копирует положительные элементы из одного одномерного целочисленного массива в другой.

Используя данную функцию, требуется выполнить копирование положительных элементов двух исходных массивов A и B в массив C.

Размер исходных массивов A и B ввести с клавиатуры. Массивы A и B заполняются случайными числами в диапазоне от -100 до 300.

Сформированный массив C вывести на экран.

Для разработанного приложения определить значение метрик Джилба на основе лексического анализа исходного кода.

1.2.2 Реализация задания

Пример реализации приложения приведен в таблице 1

Таблица 1 - Пример реализации приложения для копирования элементов массива

Номер строки	Строка приложения
1	using System;
2	class Program
3	{
4	public static int[] Copy(int[] a)
5	{
6	int[] b = new int[a.Length];
7	int j = 0;
8	for (int i = 0; i < a.Length; i++)
9	{
10	if (a[i] >= 0) { b[j] = a[i]; j++;}
11	}
12	return b;
13	}
14	
15	public static void Zapoln(ref int[] a, Random g)
16	{
17	for (int i = 0; i < a.Length; i++)
18	{
19	a[i] = g.Next(-100, 300);
20	}
21	}
22	
23	public static void Print(int[] a, string str, string str1)
24	{
25	Console.WriteLine(str);
26	for (int i = 0; i < a.Length; i++)
27	{
28	if (a[i] != 0) Console.Write(str, a[i]);
29	}
30	Console.WriteLine();
31	}
32	

```
33  static void Main(string[] args)
34  {
35    int[] a, b, c, p;
36    Random g = new Random();
37    char r;
38    do
39    {
40      Console.Clear();
41      Console.WriteLine("Определите размер первого массива");
42      a = new int[int.Parse(Console.ReadLine())];
43      Console.WriteLine("Определите размер второго массива");
44      b = new int[int.Parse(Console.ReadLine())];
45      c = new int[a.Length + b.Length];
46      Zapoln(ref a, g);
47      Zapoln(ref b, g);
48      Print(a, "{0,5}", "Первый исходный массив");
49      Print(b, "{0,5}", "Второй исходный массив");
50      p = Copy(a);
51      Array.Copy(p, 0, c, 0, a.Length);
52      p = Copy(b);
53      Array.Copy(p, 0, c, a.Length, b.Length);
54      Print(c, "{0,5}", "Результирующий массив");
55      Console.WriteLine();
56      Console.WriteLine("Выполнить повторение программы? Y/N");
57      r = char.Parse(Console.ReadLine());
58    } while (r == 'Y' || r == 'y');
59  }
60  }
```

В таблице 2 приведены операторы и операции, используемые в программе.

Таблица 2 – Словарь операторов и операций программы

№ П/п	Операторы, операции	Номера строк	Количество повторений
1	using …	1	1
2	class …	2	1
3	public static…	4,15,23,23	4
4	int[]	4,6,15,23,35	5
5	new	6,36,42,44,45	5
6	int	7,8,15,17,26	5
7	string	23,23	2
8	char	37	1
9	Random	15,36	2
10	.Length	17,6,8,26,45,45,51,53	8
11	.Next	19	1
12	Console.WriteLine()	25,30,41,43,55,56	6
13	Console.Write()	28	1
14	Console.ReadLine()	42,44,57	3
15	for ()	8,17,26	3
16	do{} while ()	38-58	1
17	if	10,28	2

18	Console.Clear()	40	1
19	Zapoln()	46,47	2
20	.Parse	42,44,57	3
21	Print()	48,49,54	3
22	Copy()	50,52	2
23	Array.Copy()	51,53	2
24	=	6,7,8,10,17,19,26,36,42,44,45, 50,52,57	14
25	>=	10	1
26	!=	28	1
27	= =	58,58	2
28	<	8,17,26	3
29	++	8,10,17,26	4
30	return	12	1
31	[ ]	4,4,6,6,10,10,10,15,19,23,28,28, 35, 42,44,45	16
32	()	4,8,10,15,17,19,23,25,26,28, 28, 30,33,36,40,41,42,42,43,44, 44,46,47,48,49,50,51,52,53,54, 55,56,57,58	35
33	{}	3(60),5(13),9(11),16(21),18(20), 24(31),27(29),34(59),39(58),10, 48,49,54	13
34	,	15,19,23,23,28,35,35,35,46,47, 48,48,49,49,51,51,51,51,53,53,53, 53,54,54,54	25
35	;	1,6,7,8,8,10,10,12,17,17,19,25,26, 26,38,30,35,36,37,40,41,42,43,44, 45,46,47,48,49,50,51,52,53,54,55, 56, 57,58	38
36	.	6,8,17,19,26,28,30,40,41,42,42,43, 44,44,45,45,46,47,48,49,50,51,51, 52,53,53,53,54,55,56,57,57	32
37	int[…]	6,42,44,45	4
38	Random()	36	1
39	“ ”	41,43,48,48,49,49,54,56	8
40	‘ ‘	58,58	2
Всего			263

Определим значения характеристик L_{IF}, L_{Loop}, L_{mod}, f, L, Cl и cl.

Значение характеристики L_{IF} определяется количеством используемых в программе операторов if . В представленном решении их два (см. таблицу 2, пункт 17).

Значение характеристики L_{Loop} определяется количеством используемых в программе циклов. В исходном тексте данной программы содержится четыре цикла: три оператора for и один do … while (см. таблицу 2, пункт 15 и 16).

Значение характеристики L_{mod} определяется количеством используемых в программе модулей в решении. В представленном решении используется четыре программных модуля, каждый из которых оперделяется следующими строками:

- static void Main().См. таблицу 1, строка 33;
- public static int[] Copy(int[] a).См. таблицу 1, строка 4;
- public static void Zapoln(ref int[] a, Random g). См. таблицу 1, строка 15;
- public static void Print(int[] a, string str, string str1) См. таблицу 1, строка 23;

Общее количество операторов условия 6, из них 2 оператора if и четыре оператора цикла. Общее число всех используемых операторов L=263 (см. таблицу 2).

Таким образом:

CL=6 – абсолютная сложность программы;

Cl=CL/L=6/263=0,0228.

Количество связей между модулями Nsv равно трем – по одной связи между основным и каждым из дополнительных модулей. Одношение числа связей к числу модулей определяется следующим образом:

$$f = \frac{N_{SV}^4}{L_{\text{mod}}} = \frac{3^4}{4} = \frac{81}{4} = 20{,}25$$

Из полученных результатов анализа текста программы следует, что исходный код имеет невысокую сложность, так как на 263 оператора текста приходится всего лишь 6 операторов условий. Общее число программных модулей решения также невелико (4 модуля), что подтверждает низкий уровень сложности программы.

2 Метрика Чепина

2.1. Теоретическое введение

Мерой сложности понимания программ на основе входных и выходных данных является метрика Н. Чепина (Ned Chapin). Смысл метода, который предложил Чепин, состоит в оценке информационной прочности отдельно взятого программного модуля на основе результатов анализа характера использования переменных, входящих в состав списка ввода и вывода.

Существует несколько модификаций метрики Чепина. Рассмотрим наиболее простой, но с точки зрения практического использования достаточно эффективный вариант этой метрики.

Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

- *P* - вводимые переменные для расчетов и для обеспечения вывода. Примером такой переменной может служить используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы - в этом случае сама переменная не модифицируется, а применяется только как контейнер для исходной информации;
- *M* - модифицируемые, или создаваемые внутри программы, переменные. К таким переменным относится большинство традиционно применяемых переменных, декларируемых в программных модулях;
- *C* - переменные, участвующие в управлении работой программного модуля (управляющие переменные). Такие переменные предназначены для передачи управления, изменения логики вычислительных процессов и т. д.;
- T - не используемые в программе (так называемые паразитные) переменные. Такие переменные не принимают непосредственного участия в реализации процесса обработки информации, ради которого написана анализируемая программа, однако они заявлены в программном модуле. Такие переменные могут использоваться для выполнения промежуточных действий и не играют принципиальной роли в решении основной задачи.

В качестве особенности применения данной метрики следует назвать необходимость учета каждой переменной в каждой функциональной группе, поскольку каждая переменная может выполнять одновременно несколько функций.

Исходное выражение для определения метрики Чепина записывается в следующем виде:

$$Q = a_1*P + a_2*M + a_3*C + a_4*T \qquad (3)$$

где a_1, a_2, a_3 и a_4, - весовые коэффициенты.

Весовые коэффициенты в расчетном выражении значения метрики применяются для отражения различного влияния на сложность программы каждой функциональной группы переменных. По мнению автора метрики наибольший вес, равный 3, должна иметь функциональная группа С, так как она непосредственно влияет на поток управления

программы. Весовые коэффициенты остальных групп Чепин распределяет следующим образом:

- $a_1 = 1$;
- $a_2 = 2$;
- $a_4 = 0{,}5$;

Примечательно, что весовой коэффициент группы *T* не равен 0, несмотря на то, что часть используемых переменных может не применяться в программе. Это объясняется тем, что «паразитные» переменные сами по себе не увеличивают сложность потока данных программы, но очень часто затрудняют ее понимание.

С учетом весовых коэффициентов расчетное выражение метрики Чепина приобретает следующий вид:

$Q = P+2*M + 3*C + 0{,}5*T.$

Следует отметить, что метрика сложности программы Чепина также основана на анализе исходных текстов программ, что обеспечивает единый подход к автоматизации их расчета и может быть рассчитана с использованием специально разработанного программного анализатора.

2.2. Пример: «Простые числа в матрице»

Дана целочисленная матрица размером *N*M*. Вычислить и записать в одномерный массив количество простых чисел в каждом столбце матрицы. Размерность матрицы задается с клавиатуры, заполнение матрицы осуществляется посредством датчика случайных чисел. Разработать программу для решения задачи. На основе лексического анализа исходного текста программы определить значение метрики Чепина.

2.2.1 Реализация программы

Текст программы для реализации возможного алгоритма решения поставленной задачи представлен в таблице 3.

Таблица 3- Текст программы

Номера строк	Строки программы
1	using System;
2	namespace chapin
3	{
4	class Program
5	{
6	static void Main(string[] args)
7	{
8	int n, m;
9	
10	int[,] a;
11	int[] b;
12	ConsoleKeyInfo клавиша;
13	int i, j, k, d;
14	bool p = false;
15	Random g;
16	do
17	{
18	Console.Clear();
19	Console.Write("Введите количество строк");
20	n = int.Parse(Console.ReadLine());
21	Console.Write("Введите количество столбцов");
22	m = int.Parse(Console.ReadLine());
23	g = new Random();

24	a = new int[n, m];
25	for (i = 0; i < n; i++)
26	for (j = 0; j < m; j++)
27	{
28	a[i,j] = g.Next(0,101);
29	}
30	
31	Console.WriteLine("\nИсходная матрица");
32	for (i = 0; i < n; i++, Console.WriteLine())
33	for (j = 0; j < m; j++)
34	Console.Write("{0,8:d}", a[i, j]);
35	b = new int[m];
36	
37	for (j = 0; j < m; j++)
38	{
39	for (i = k = 0; i < n; i++)
40	{
41	p = true;
42	for (d = 2; d < a[i, j]; d++)
42	if (a[i, j] % d == 0) p = false;
44	
45	
46	if (p) k++;
47	b[j] = k;
48	}
49	}
50	
51	
52	Console.WriteLine("\nКоличество простых чисел");
53	for (i = 0; i < b.Length; i++)
54	Console.Write("{0,8:d}", b[i]);
55	
56	Console.WriteLine("\nДля выхода нажмите клавишу ESC");
57	клавиша = Console.ReadKey(true);
58	} while (клавиша.Key != ConsoleKey.Escape);
59	}
60	}
61	}

2.2.2 Оценка характеристик программы

Выполним оценку качества программы с помощью метрики Чепина, которая позволяет оценить меру трудности понимания программы на основе входных и выходных данных. Все множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

- *P* - вводимые переменные для расчетов и для обеспечения вывода;
- *M* - модифицируемые, создаваемые внутри программы переменные;
- C - переменные, участвующие в управлении работой программного модуля (управляющие переменные);
- *T* - не используемые в программе переменные.

В таблице 4 приведен анализ исходного текста программы

Таблица 4 – Анализ исходного кода программы

№ п/п	Наименование переменных	Номера строк
P (для расчетов и для обеспечения вывода)		
1	*m*	8
2	*n*	8
3	*a*	10
M (модифицируемые или создаваемые внутри программы переменные)		
1	*i*	13
2	*j*	13
3	*k*	13
4	*d*	13
5	*b*	11
C (управляющие переменные)		
1	*g*	15
2	*p*	14
3	*клавиша*	12
T (не используемые в программе переменные)		
	Отсутствуют	

Переменные *m,n* и *a* используются в качестве исходных данных.

Переменные ***i, j, k, d*** и *b* в процессе выполнения программы создаются и модифицируются.

Переменные *g, p* и *клавиша* используются для управления выполнением программы.

Таким образом, исходя из результатов анализа исходного текста программы, получаем следующие значения характеристик:

- *P* = 3 - количество переменных для расчетов;

M = 5 - количество модифицируемых переменных;

- C = 3 - количество переменных, используемых в управлении программой;
- *T* = 0 - количество неиспользуемых переменных (такие переменные в программе отсутствуют).

Расчет метрики Чепина:

$Q = P + 2M + 3\text{-}C + 0{,}5T = 3 + 2\text{-}5 + 3\text{-}3 + 0{,}5\text{-}0 = 22$.

На основе полученных значений метрики Чепина уровень сложности данного решения можно считать сравнительно низким, как так в исходном тексте программы используется незначительное количество переменных, что не затрудняет понимание программы.

Наряду с рассмотренными используются и другие метрики качества программного обеспечения, например:

- метрики стилистики и понятности программ;
- метрика уровня комментированности программ;
- метрика изменения длины программной документации и т. д.

Порядок выполнения работы

Задание 1.

Набрать и проверить работу программ из теоретической части. Разобраться в подсчете метрик Джилба и Чепина.

Задание 2.

1. Напишите программу согласно вашему варианту.
2. Оцените качество разработанной программы по методике Джилба.

Заполните таблицу

Анализ исходного кода согласно метрике Джилба

№ П/п	Операторы, операции	Номера строк	Количество повторений
…	…	…	…

3. Оцените качество разработанной программы по методике Чепина.
   Заполните таблицу
   Анализ исходного кода согласно метрике Чепина

№ п/п	Наименование переменных	Номера строк
P (для расчетов и для обеспечения вывода)		
1		
2	...	…
M (модифицируемые или создаваемые внутри программы переменные)		
1		
2	...	…
C (управляющие переменные)		
1		
2	...	…
T (не используемые в программе переменные)		
1	…	…

4. Оптимизируйте программу, если это возможно.
5. Оцените качество оптимизированной программы по методикам Джилба и Чепина.
6. *Оценить эффективность созданных программ. Сделайте выводы.*

Варианты:

1. Даны две точки A(x1,y1) и B(x2,y2). Составить программу, определяющую, которая из точек находится ближе к началу координат.
2. Даны действительные числа x и y, не равные друг другу. Меньшее из этих двух чисел заменить половиной их суммы, а большее - их удвоенным произведением.
3. Даны целые числа in, n. Если числа не равны, то заменить каждое из них тем же числом, равным большему из исходных, а если равны, го заменить числа нулями.
4. Определить, делителем каких чисел a, b. c является число k.
5. Услуги телефонной сети оплачиваются по следующему правилу: за разговоры до A минут в месяц - B рублей, а разговоры сверх установленной нормы оплачиваются из расчета C рублей за минуту. Написать программу, вычисляющую плату за пользование телефоном для введенного времени разговоров за месяц.
6. **Грузовой автомобиль выехал из одного города в другой со скоростью V1 км/ч. Через t ч в этом же направлении выехал легковой автомобиль со скоростью V2 км/ч. Составить программу, определяющую догонит ли легковой автомобиль грузовой через t, ч после своего выезда.**
7. Определить правильность даты, введенной с клавиатуры (.число - от 1 до 31, месяц - от 1 до 12). Если введены некорректные данные, то сообщить об этом.
8. Составить программу, определяющую результат гадания на ромашке - «любит - не любит», взяв за исходное данное количество лепестков n.
9. **Рис расфасован в два пакета. Масса первого - m кг. второго - n кг. Составить программу. определяющую:**
   a. какой пакет тяжелее - первый или второй;
   b. массу более тяжелого пакета.
10. Написать программу, которая анализирует данные о возрасте и относит человека к одной из четырех групп: дошкольник, ученик, работник, пенсионер. Возраст вводится с клавиатуры.
11. Написать программу нахождения суммы большего и меньшего из трех чисел.
12. **На оси OX расположены три точки a, b, c. Определить, какая из точек b или c**

расположена ближе к точке a.

13. Написать программу решения уравнения ax3 + b x = 0 для произвольных a, b.

14. Заданы размеры A, B прямоугольного отверстия и размеры x, y,z кирпича. Определить, пройдет ли кирпич через отверстие.

15. Подсчитать количество отрицательных и положительных чисел среди чисел a, b, c.