МЕТРИКИ ЛОРЕНЦА И КИДДА

Теоретические сведения

Комплексный набор метрик, предложенный в 1994 г. М. Лоренцем и Д. Киддом (M. Lorenz и J. Kidd), имеет практическую направленность на использование в промышленной разработке программного обеспечения.

Набор включает 10 метрик, которые, в свою очередь, классифицируют в следующие группы:

- метрики размера, основанные на подсчете свойств и операций для отдельных классов, а также их средних значений для всей объектно-ориентированной системы;
- метрики наследования, учитывающие способы повторного использования операций в иерархии классов;
- внутренние метрики, отвечающие на вопросы связности и кодирования;
- внешние метрики, изучающие сцепление и повторное использование.

Размер класса CS (Class Size) - общий размер класса определяется на основании определения следующих показателей:

- общее количество операций;
- количество свойств.

Указанные измерения в обоих случаях следует проводить с учетом частных (приватных) и наследуемых экземплярных операций, которые инкапсулируются внутри класса:

$$CS = C_\Sigma + S_\Sigma,$$

где C_Σ - количество инкапсулированных классом методов (операций); S_Σ - количество инкапсулированных классом свойств.

CS может определяться взвешенной суммой C_Σ и S_Σ Следует заметить, что метрика WMC Чидамбера и Кемерера также является взвешенной метрикой размера класса.

Большие значения метрики CS указывают на то, что класс имеет слишком много обязанностей, что уменьшает возможность повторного использования класса, усложняет его реализацию и тестирование.

При определении размера класса больший удельный вес придают унаследованным (публичным) операциям и свойствам, потому что

приватные операции и свойства обеспечивают специализацию и являются более локализованными в проекте.

Могут вычисляться средние количества свойств и операций класса. Чем меньше среднее значение размера CS, тем больше вероятность повторного использования класса. Рекомендуемое значение метрики CS ограничено сверху 20 инкапсулированными методами и свойствами ($CS < 20$).

Количество операций, переопределяемых подклассом NOO (Number of Operations Overridden by a Subclass). Переопределением называют случай, когда подкласс замещает операцию, унаследованную от суперкласса, своей собственной версией.

Большие значения NOO указывают на возникшие проблемы проектирования. Понятно, что подкласс должен расширять операции суперкласса, что проявляется в виде новых имен операций. Если же значение метрики NOO достаточно велико, то это означает нарушение разработчиком абстракции суперкласса. Это явление ослабляет иерархию классов, усложняет тестирование и модификацию программного обеспечения. Рекомендуемое значение метрики NOO составляет три метода.

Количество операций, добавленных подклассом NOA (Number of Operations Added by a Subclass), определяется количеством добавленных относительно родительского класса собственных методов (операций):

$$NOA = N_{\Sigma},$$

где N_{Σ} - количество новых методов класса, добавленных относительно суперкласса.

С увеличением NOA подкласс приобретает меньшую общность со своим суперклассом, что требует больших трудозатрат по тестированию и внесению изменений. При увеличении высоты дерева иерархии классов (с ростом значения DIT должно уменьшаться количество новых методов классов нижних уровней. Для рекомендуемых граничных значений размера класса ($CS = 20$) и высоты дерева иерархии классов ($DIT = 6$) значение NOA ограничено значением 4).

Индекс специализации SI (Specialization Index) характеризует грубую оценку степени специализации каждого подкласса при добавлении, удалении или переопределении операций:

$$SI = \frac{NOO \cdot u}{M_{общ}},$$

где u - номер уровня в иерархии, на котором находится подкласс; $M_{общ}$- общее количество методов класса.

Чем выше значение метрики SI, тем выше вероятность того, что в иерархии классов есть отдельные экземпляры, нарушающие абстракцию суперкласса. Рекомендуемое значение показателя SI ограничено сверху величиной 0,15, т. е. $SI < 0{,}15$.

Следующая группа метрик предназначена для оценки операций в классах. Как правило, методы бывают небольшими как по размеру, так и по логической сложности. В то же время истинные характеристики операций помогают более глубоко понять особенности создаваемой системы.

Средний размер операции AOS (Average Operation Size) определяется количеством сообщений, порождаемых операцией. В качестве оценки размера может использоваться количество строк программы, однако LOC-оценки приводят к известным проблемам. Иным вариантом может являться количество сообщений, посланных операцией.

Рост значения данного показателя означает, что обязанности размещены в классе не очень удачно. Рекомендуемое значение метрики AOS не должно превышать 9. Увеличение среднего размера относительно этой границы рассматривают как показатель неудачного проектирования обязанностей класса.

Сложность операции OC (Operation Complexity) может быть вычислена на основе стандартных метрик сложности (например, с помощью LOC или FP -оценок, метрики цикломатической сложности, метрики Холстеда). Лоренц и Кидд предложили вычислять значение OC суммированием оценок с весовыми коэффициентами, приведенными в табл. 1.

Рекомендуемое значение метрики ограничено числом 65 ($OC < 65$).

Таблица 1

Действие	Вес
Определение (описание) переменной-параметра	0,3
Определение (описание) временной переменной	0,5
Присваивание значения	0,5
Вложенное выражение	0,5
Сообщение без параметров	1
Арифметическая операция	2
Сообщение с параметрами	3
Вызов стандартной функции интерфейса (*API*)	5
Вызов пользовательской функции (простой вызов)	7

Среднее количество параметров на операцию ANP (Average Number of Parameters per operation) определяется отношением числа параметров к количеству операций (методов) класса.

$$ANP = \frac{\text{Количество\_параметров}}{\text{Количество\_операций}}.$$

Чем больше параметров у операции, тем сложнее взаимодействие между объектами. Поэтому значение метрики ANP должно быть как можно меньшим. Рекомендуемое значение $ANP = 0.7$.

Следующая группа метрик предназначена для оценки показателей процесса разработки ПС. Центральным вопросом в процессе разработки является прогноз размера создаваемого продукта.

Количество описаний сценариев NSS (Number of Scenario Scripts) измеряется или количеством классов, реализующих требования к ПО, или количеством состояний для каждого класса, или количеством методов класса. При своем не совсем обычном способе измерения метрика NSS является достаточно эффективным индикатором размера создаваемой программы. Рекомендуется не менее одного сценария. Рост количества сценариев неминуемо ведет к увеличению размера программы.

Количество ключевых классов NKC (Number of Key Classes) адекватно характеризует предстоящий объем работы по программированию. Ключевой класс прямо связан с проблемной областью, для которой предназначена система, поэтому, если предметная область специфична, то возможность повторного использования существующего ключевого класса маловероятна и требуется самостоятельная разработка «с нуля». Авторы этого набора метрик полагают, что в типовой объектно-ориентированной системе на долю ключевых классов приходится от 20 до 40% общего количества классов. Остальные классы реализуют общую инфраструктуру (интерфейсы, коммуникации, базы данных). Рекомендуется ограничивать значение метрики снизу значением 0,2. Если значение метрики $NKC < 0.2$ от общего количества классов системы, следует пересмотреть выделение классов.

Количество подсистем $NSUB$ (Number of subsystem) определяется непосредственным подсчетом. Количество подсистем обеспечивает понимание таких вопросов, как размещение ресурсов, планирование (с акцентом на параллельной разработке), общие затраты на интеграцию. Рекомендуется выделять в программном комплексе (системе) не менее

трех подсистем (т. е. значение $NSUB > 3$). Количество подсистем характеризует трудоемкость и управляемость проекта.

Значения метрик последней группы целесообразно накапливать по мере выполнения проектов по разработке ПС. Эти данные могут применяться для вычисления показателей производительности. Совместное применение метрик позволяет оценивать необходимые ресурсы на выполнение проекта: затраты, продолжительность разработки, численность привлекаемого персонала и другие характеристики.

Задача «Платеж за электроэнергию»

Необходимо определить класс, описывающий платеж за электроэнергию. В рамках класса следует предусмотреть следующие поля:

- фамилия плательщика;
- потребление электроэнергии за оплачиваемый месяц;
- нормативное среднемесячное потребление;
- тариф (стоимость одного киловатт-часа).

Рекомендуется применять следующие методы:

- вычисление суммы оплаты;
- формирование сводной информации по одному платежу (вид платежа, фамилия, сумма, потребление).

Платеж может выполняться по показаниям счетчика или по нормативно установленному среднемесячному потреблению. В процессе эксплуатации программы тариф и нормативно установленное среднемесячное потребление не изменяются. Для создания конкретного платежа предусмотреть соответствующий конструктор.

Все платежи должны сохраняться в архиве. Запросы по ведению архива выполняются статическими методами класса «Запрос»:

- занесение платежей в архив. Данные платежа вводятся с клавиатуры. Ввод отрицательного показания счетчика означает оплату по нормативно установленному среднемесячному потреблению;
- вывод сводной информации из архива.

Архив моделируется массивом объектов.

В основном классе «Платежи» следует сформировать архив платежей. По данным архива предусмотреть выдачу сводной информации о платежах.

При решении задачи необходимо разработать исходный код программы, а также определить оценки характеристик программы на основе объектно-ориентированных метрик Чидамбера и Кемерера.

Реализация программы

Текст программы на языке C# для реализации возможного алгоритма решения поставленной задачи представлен на рис. 1.

Номера строк	Строки программы
1	using System;
2	using System.Collections.Generic;
3	using System.Text;
4	namespace EX1
5	{
6	class Электро
7	{
8	private static double тариф = 1.84;
9	private static int потреблениеСреднее = 300;
10	private string фамилия;
11	private int потребление;
12	public Электро(string фамилия)
13	{
14	this.фамилия = фамилия;
15	потребление = потреблениеСреднее;
16	}
17	public Электро(string фамилия, int текущее, int предыдущее)
18	{
19	this.фамилия = фамилия;
20	потребление = текущее - предыдущее;
21	}
22	public double Сумма()
23	{
24	return потребление * тариф;
25	}
26	public string Инфо()
27	{
28	return string.Format("{0,-20}{1,-20}{2,10:f2}{3,10:d6}",
29	"Электроэнергия",фамилия,Сумма(),потребление);
30	}
31	}
32	class Запрос
33	{
34	public static void Заполнить(Электро[] платеж)
35	{
36	string фам;
37	int счПред=0, счТек=0;
38	Console.Clear();
39	for (int i = 0; i < платеж.Length; i++)
40	{
41	Console.Write("Платеж "+ i + ": Фамилия -> ");
42	фам = Console.ReadLine();
43	Console.Write("Платеж " + i + ": Текущее значение счетчика -> ");
44	счТек = int.Parse(Console.ReadLine());
45	if (счТек > 0)
46	{
47	Console.Write("Платеж " + i + ": Предыдущее значение счетчика -> ");

```
48      счПред = int.Parse(Console.ReadLine());
49      }
50      if(счТек <= 0)
51       платеж[i] = new Электро(фам);
52      else
53       платеж[i] = new Электро(фам, счТек, счПред);
54      }
55     }
56     public static void Вывести(Электро[] платеж)
57     {
58      for (int i = 0; i < платеж.Length; i++)
59      Console.WriteLine(платеж[i].Инфо());
60     }
61     }
62     class Платежи
63     {
64      static void Main(string[] args)
65      {
66      ConsoleKeyInfo rep;
67      Электро[] плэ;
68      int кпэ;
69      do
70      {
71      Console.Clear();
72      Console.Write("Количество платежей за электроэнергию: ");
73      кпэ = int.Parse(Console.ReadLine());
74      плэ = new Электро[кпэ];
75      Запрос.Заполнить(плэ);
76      Запрос.Вывести(плэ);
77      Console.WriteLine("Для выхода нажмите ESC");
78      rep = Console.ReadKey(true);
79      }while(rep.Key!= ConsoleKey.Escape);
80      }
81      }
82     }
```

Рис. 1. Пример реализации программы «Платеж за электроэнергию»

Оценка характеристик программы.

Метрика размера класса *cs* определяется на основание следующих показателей:

- общее количество операций;
- количество свойств,

а значение рассчитывается по формуле

$$CS = C_\Sigma + S_\Sigma,$$

где C_Σ - количество инкапсулированных классом методов; S_Σ - количество инкапсулируемых классом свойств (полей классов).

Для класса Электро число инкапсулированных методов $C_\Sigma = 4$, количество инкапсулированных полей $S_\Sigma = 4$ (см. рис. 4.1):

- private static double тариф = 1.84;
- private static int потреблениеСреднее = 300;
- private string фамилия',
- private int потребление.

Таким образом, для класса *Электро*

$$CS = C_\Sigma + S_\Sigma = 4 + 4 = 8.$$

Для класса Запрос количество инкапсулируемых методов $C_\Sigma = 2$, количество инкапсулированных полей $C_\Sigma = 0$, так как в классе не определены поля:

$$CS = C_\Sigma + S_\Sigma = 2 + 0 = 2.$$

Для класса Платежи количество инкапсулируемых методов $C_\Sigma = 1$, количество инкапсулированных полей $C_\Sigma = 0$, так как в классе не определены поля.

$$CS = C_\Sigma + S_\Sigma = 1 + 0 = 1.$$

Полученные значения метрик классов не превышают значения 20, следовательно, сложность классов не превышает требуемый уровень и разбиение программы на дополнительные модули не требуется.

Количество операций, переопределяемых подклассом NOO, количество операций, добавленных подклассом NOA, индекс специализации SI для данного решения не определяются, так как в исходном коде программы не реализован принцип наследования.

Метрика среднего размера операции AOS определяется количеством сообщений, порождаемых операцией. В качестве посылаемого методом сообщения будем рассматривать обращение к методам других классов. Определим значение метрики для классов. В классе Электро обращения к методам осуществляются только в методе $public\ string$ Инфо() . Количество посылаемых сообщений равно 2:

- обращение к методу $string.FormatQ$;
- обращение к методу Сумма().

Таким образом, для класса Электро $AOS = 2$.

В классе Запрос обращений к методам наблюдается 12 раз (см. определение метрики CBO(Запрос)). Для класса Запрос $AOS = 12$.

В классе Платежи встречается 10 обращений к методам (из определения метрики CBO(Платежи)), следовательно, для этого класса $AOS = 10$. Из полученных результатов следует, что наиболее сложным является класс Запрос.

Определим сложность операции ОС, которую будем вычислять по количеству строк в коде метода. При вычислении воспользуемся весомыми коэффициентами, предложенными Лоренцом и Киддом (табл. 2).

Таблица 2

Весовые коэффициенты

Действие	Вес
Определение (описание) переменной-параметра	0,3
Определение (описание) временной переменной	0,5
Присваивание значения	0,5
Вложенное выражение	0,5
Сообщение без параметров	1
Арифметическая операция	2
Сообщение с параметрами	3
Вызов стандартной функции интерфейса (*API*)	5
Вызов пользовательской функции (простой вызов)	7

Значения весовых коэффициентов зависят от количества резервируемой памяти, необходимой для размещения исполняемых кодов операций программы. Наименьшее количество кодовых операций требуют операторы описания параметров методов. Операторы описании переменных и массивов, а также операторы присваивания требуют большего количества машинных кодов. Арифметические операции, операции обмена данными требуют еще большего объема памяти, так как кроме исполняемых машинных кодов требуется дополнительная память и для обрабатываемых данных, которые необходимо передавать из одной функции обработки в другую.

Определим количество строк по предложенным разновидностям действий для класса Электро, которые для удобства сведем в таблицы, рассматривая применяемые методы.

Таблица 3

Метод *Public* Электро (*String* фамилия)

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	0
Определение (описание) временной переменной	0,5	0
Присваивание значения	0,5	2
Вложенное выражение	0,5	0
Сообщение без параметров	1	0
Арифметическая операция	2	0
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	0
Вызов пользовательской функции (простой вызов)	7	0

Расчет сложности операции ОС дает следующий результат: $ОС = 0{,}5 * 2 = 1$.

Таблица 4

Метод

public Электро (*String* фамилия, *int* текущее, *int* предыдущее)

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	0
Определение (описание) временной переменной	0,5	0
Присваивание значения	0,5	2
Вложенное выражение	0,5	0
Сообщение без параметров	1	0
Арифметическая операция	2	1
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	0
Вызов пользовательской функции (простой вызов)	7	0

Значение метрики сложности операции $OC = 0.5 * 2 + 2 * 1 = 3$.

Таблица 5

Метод *public double* Сумма()

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	0
Определение (описание) временной переменной	0,5	0
Присваивание значения	0,5	0
Вложенное выражение	0,5	0
Сообщение без параметров	1	0
Арифметическая операция	2	1
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	0
Вызов пользовательской функции (простой вызов)	7	0

Метрика сложности операции для данного метода ОС = 2*1=2.

Таблица 6

Метод *Public String* Инфо()

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	0
Определение (описание) временной переменной	0,5	0
Присваивание значения	0,5	0
Вложенное выражение	0,5	0
Сообщение без параметров	1	0
Арифметическая операция	2	0
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	1
Вызов пользовательской функции (простой вызов)	7	1

Метрика сложности операции для данного метода:

$$OC = 5 * 1 + 7 * 1 = 12.$$

Из полученных значений метрики сложности операции можно сделать вывод, что все операции класса Электро имеют сложность низкого уровня, которая не превышает критическое значение (менее 65).

Определим показатели сложности операции ОС для класса Запрос.

Таблица 7

Метод *public static void* Заполнить (Электро[] платеж)

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	1
Определение (описание) временной переменной	0,5	4
Присваивание значения	0,5	8
Вложенное выражение	0,5	0
Сообщение без параметров	1	0
Арифметическая операция	2	4
Сообщение с параметрами	3	3
Вызов стандартной функции интерфейса (*API*)	5	10
Вызов пользовательской функции (простой вызов)	7	2

Метрика сложности операции для данного метода:

$$OC = 0.3 * 1 + 0.5 * 3 + 0.5 * 8 + 2 * 4 + 3 * 3 + 5 * 10 + 7 * 2 = 87.3$$

Таблица 8

Метод *public static void* Вывести (Электро[] платеж)

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	1
Определение (описание) временной переменной	0,5	1
Присваивание значения	0,5	1
Вложенное выражение	0,5	0
Сообщение без параметров	1	1
Арифметическая операция	2	1
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	1
Вызов пользовательской функции (простой вызов)	7	1

Метрика сложности операции определится следующим образом:

$$OC = 0.3 * 1 + 0.5 * 1 + 0.5 * 1 + 1 * 1 + 2 * 1 + 5 * 1 + 7 * 1 = 16.3$$

Из полученных значений видно, что одна из операций класса Запрос имеет значительный уровень сложности (значение метрики ОС превышает 65).

Определим характеристику ОС для класса Платежи.

Таблица 9

Метод static void Main(string[] args)

Действие	Вес	Количество строк
Определение (описание) переменной-параметра	0,3	1
Определение (описание) временной переменной	0,5	3
Присваивание значения	0,5	3
Вложенное выражение	0,5	1
Сообщение без параметров	1	2
Арифметическая операция	2	0
Сообщение с параметрами	3	0
Вызов стандартной функции интерфейса (*API*)	5	6
Вызов пользовательской функции (простой вызов)	7	2

Метрика сложности операции для данного метода:

$$OC = 0.3 * 1 + 0.5 * 3 + 0.5 * 3 + 0.5 * 1 + 1 * 2 + 5 * 6 + 7 * 2 = 49.8$$

В связи с тем что сложность операции класса Платежи не превышает 65, можно сделать вывод о допустимом уровне сложности данного метода.

Определим среднее количество параметров на операцию ANP ($Average\ Number\ of\ Parameters\ per\ operation$), которое рассчитывается по следующему соотношению:

$$ANP = \frac{\text{Количество\_параметров}}{\text{Количество\_операций}}.$$

Количество параметров в классе Электро четыре: один параметр в методе $public$ Электро($string$ фамилия), три параметра в методе $public$ Электро($string$ фамилия, int текущее, int предыдущее), остальные методы входных параметров не имеют. Количество методов в классе четыре, следовательно,

$$ANP = 4 / 4 = 1.$$

Количество параметров в классе Запрос два: один - в методе $public\ static\ void$ Заполнитъ(Электро[ ] платеж), другой - в методе $public\ static\ void$ Вывести(Электро[ ] платеж). Количество методов - два, отсюда следует, что

$$ANP = 2 / 2 = 1.$$

Параметры в классе Платежи отсутствуют, в классе имеется один метод, поэтому

$$ANP = 0 / 2 = 0.$$

Среднее количество параметров на операцию в большинстве классов свыше допустимого (более 0,7), что говорит о низком качестве программы.

Определим количество сценариев NSS $(Number\ of\ Scenario\ Scripts)$, опираясь на количество методов в классе. Для класса Электро значение $NSS = 4$, для класса Запрос $NSS = 2$, для класса Платежи показатель $NSS = 1$.

Количество сценариев для исследуемой программы невелико, что объясняет сравнительно небольшие размеры программы.

Количество ключевых классов NKC $(Number\ of\ Key\ Classes)$ в рассматриваемой программе равно трем. Все классы, определенные в программе, можно назвать ключевыми, так как они напрямую связаны с проблемной областью поставленной задачи. Таким образом, $NKC = 1$, т. е. количество ключевых классов от общего количества классов программы составляет 100%.

В рассматриваемом решении можно выделить три подсистемы:

- класс, определяющий элемент платежа как объект;
- класс, определяющий операции создания объектов и вывода информации по ним;
- основной управляющий класс.

Таким образом, количество подсистем $NSUB$ $(Number\ of\ subsystem) = 3$. Количество подсистем программы обеспечивает достаточную управляемость проекта и невысокую трудоемкость