Лекция 7. Метрики объектно-ориентированных программных систем

При конструировании объектно-ориентированных программных систем значительная часть затрат приходится на создание визуальных моделей. Очень важно корректно и всесторонне оценить качество этих моделей, сопоставив качеству числовую оценку. Решение данной задачи требует введения специального метрического аппарата. Такой аппарат развивает идеи классического оценивания сложных программных систем, основанного на метриках сложности, связности и сцепления. Вместе с тем он учитывает специфические особенности объектно-ориентированных решений. В этом разделе обсуждаются наиболее известные объектно-ориентированные метрики, а также описывается методика их применения.

Метрические особенности объектно-ориентированных программных систем

Объектно-ориентированные метрики вводятся с целью:

- улучшить понимание качества продукта;
- оценить эффективность процесса конструирования;
- улучшить качество работы на этапе проектирования.

Все эти цели важны, но для программного инженера главная цель — повышение качества продукта. Возникает вопрос — как измерить качество объектно-ориентированной системы?

Для любого инженерного продукта метрики должны ориентироваться на его уникальные характеристики. Например, для электропоезда вряд ли полезна метрика «расход угля на километр пробега». С точки зрения метрик выделяют пять характеристик объектно-ориентированных систем: локализацию, инкапсуляцию, информационную закрытость, наследование и способы абстрагирования объектов. Эти характеристики оказывают максимальное влияние на объектно-ориентированные метрики.

Локализация

Локализация фиксирует способ группировки информации в программе. В классических методах, где используется функциональная декомпозиция, информация локализуется вокруг функций. Функции в них реализуются как процедурные модули. В методах, управляемых данными, информация группируется вокруг структур данных. В объектно-ориентированной среде информация группируется внутри классов или объектов (инкапсуляцией как данных, так и процессов).

Поскольку в классических методах основной механизм локализации — функция, программные метрики ориентированы на внутреннюю структуру или сложность функций (длина модуля, связность, цикломатическая сложность) или на способ, которым функции связываются друг с другом (сцепление модулей).

Так как в объектно-ориентированной системе базовым элементом является класс, то локализация здесь основывается на объектах. Поэтому метрики должны применяться к классу (объекту) как к комплексной сущности. Кроме того, между операциями (функциями) и классами могут быть отношения не только «один-к-одному». Поэтому метрики, отображающие способы

взаимодействия классов, должны быть приспособлены к отношениям «один-ко-многим», «многие-ко-многим».

Инкапсуляция

Вспомним, что инкапсуляция — упаковка (связывание) совокупности элементов. Для классических ПС примерами низкоуровневой инкапсуляции являются записи и массивы. Механизмом инкапсуляции среднего уровня являются подпрограммы (процедуры, функции).

В объектно-ориентированных системах инкапсулируются обязанности класса, представляемые его свойствами (а для агрегатов — и свойствами других классов), операциями и состояниями.

Для метрик учет инкапсуляции приводит к смещению фокуса измерений с одного модуля на группу свойств и обрабатывающих модулей (операций). Кроме того, инкапсуляция переводит измерения на более высокий уровень абстракции (пример — метрика «количество операций на класс»). Напротив, классические метрики ориентированы на низкий уровень — количество булевых условий (цикломатическая сложность) и количество строк программы.

Информационная закрытость

Информационная закрытость делает невидимыми операционные детали программного компонента. Другим компонентам доступна только необходимая информация.

Качественные объектно-ориентированные системы поддерживают высокий уровень информационной закрытости. Таким образом, метрики, измеряющие степень достигнутой закрытости, тем самым отображают качество объектно-ориентированного проекта.

Наследование

Наследование — механизм, обеспечивающий тиражирование обязанностей одного класса в другие классы. Наследование распространяется через все уровни иерархии классов. Стандартные ПС не поддерживают эту характеристику.

Поскольку наследование — основная характеристика объектно-ориентированных систем, на ней фокусируются многие объектно-ориентированные метрики (количество детей — потомков класса, количество родителей, высота класса в иерархии наследования).

Абстракция

Абстракция — это механизм, который позволяет проектировщику выделять главное в программном компоненте (как свойства, так и операции) без учета второстепенных деталей. По мере перемещения на более высокие уровни абстракции мы игнорируем все большее количество деталей, обеспечивая все более общее представление понятия или элемента. По мере перемещения на более низкие уровни абстракции мы вводим все большее количество деталей, обеспечивая более удачное представление понятия или элемента.

Класс — это абстракция, которая может быть представлена на различных уровнях детализации и различными способами (например, как список операций, последовательность

состояний, последовательности взаимодействий). Поэтому объектно-ориентированные метрики должны представлять абстракции в терминах измерений класса. Примеры: количество экземпляров класса в приложении, количество родовых классов на приложение, отношение количества родовых к количеству неродовых классов.

Эволюция мер связи для объектно-ориентированных программных систем

Классической мерой сложности внутренних связей модуля является связность, а классической мерой сложности внешних связей — сцепление. Рассмотрим развитие этих мер применительно к объектно-ориентированным системам.

Связность объектов

В классическом методе Л. Констентайна и Э. Йордана определены семь типов связности.

1. ***Связность по совпадению.*** В модуле отсутствуют явно выраженные внутренние связи.
2. ***Логическая связность. Части модуля объединены по принципу функционального подобия.***
3. ***Временная связность. Части модуля не связаны, но необходимы в один и тот же период работы системы.***
4. ***Процедурная связность. Части модуля связаны порядком выполняемых ими действий, реализующих некоторый сценарий поведения.***
5. ***Коммуникативная связность. Части модуля связаны по данным (работают с одной и той же структурой данных).***
6. ***Информационная (последовательная) связность. Выходные данные одной части используются как входные данные в другой части модуля.***
7. ***Функциональная связность. Части модуля вместе реализуют*** одну функцию.

Этот метод функционален по своей природе, поэтому наибольшей связностью здесь объявлена функциональная связность. Вместе с тем одним из принципиальных преимуществ объектно-ориентированного подхода является естественная связанность объектов.

Максимально связанным является объект, в котором представляется единая сущность и в который включены все операции над этой сущностью. Например, максимально связанным является объект, представляющий таблицу символов компилятора, если в него включены все функции, такие как «Добавить символ», «Поиск в таблице» и т. д.

Следовательно, восьмой тип связности можно определить так:

8. ***Объектная связность.*** Каждая операция обеспечивает функциональность, которая предусматривает, что все свойства объекта будут модифицироваться, отображаться и использоваться как базис для предоставления услуг.

Высокая связность — желательная характеристика, так как она означает, что объект представляет единую часть в проблемной области, существует в едином пространстве. При изменении системы все действия над частью инкапсулируются в едином компоненте. Поэтому для производства изменения нет нужды модифицировать много компонентов.

Если функциональность в объектно-ориентированной системе обеспечивается наследованием от суперклассов, то связность объекта, который наследует свойства и операции, уменьшается. В этом случае нельзя рассматривать объект как отдельный модуль — должны учитываться все его суперклассы. Системные средства просмотра содействуют такому учету. Однако понимание элемента, который наследует свойства от нескольких суперклассов, резко усложняется.

Обсудим конкретные метрики для вычисления связности классов.

Метрики связности по данным

Л. Отт и Б. Мехра разработали модель секционирования класса. Секционирование основывается на экземплярных переменных класса. Для каждого метода класса получают ряд секций, а затем производят объединение всех секций класса. Измерение связности основывается на количестве *лексем данных* (*data tokens*), которые появляются в нескольких секциях и «склеивают» секции в модуль. Под лексемами данных здесь понимают определения констант и переменных или ссылки на константы и переменные.

Базовым понятием методики является секция данных. Она составляется для каждого выходного параметра метода. *Секция данных* — это последовательность лексем данных в операторах, которые требуются для вычисления этого параметра.

Например, на рис. 14.1 представлен программный текст метода SumAndProduct. Все лексемы, входящие в секцию переменной SumN, выделены рамками. Сама секция для SumN записывается как следующая последовательность лексем:

$N_1 \bullet SumN_1 \bullet I_1 \bullet SumN_2 \bullet O_1 \bullet I_2 \bullet 1_2 \bullet N_2 \bullet SumN_3\ SumN_4 \bullet I_3$.

```
procedure SumAndProduct
(N : integer;
 var SumN, ProdN : integer );
 var
    I : integer;
begin
    SumN := 0 ;
    ProdN := 1;
    for I := 1 to N do begin
                          SumN := SumN + I ;
                          ProdN := ProdN * I
                       end
end;
```

Рис. 14.1. Секция данных для переменной SumN

Заметим, что индекс в «1_2» указывает на второе вхождение лексемы «1» в текст метода. Аналогичным образом определяется секция для переменной ProdN:

$N_1 \bullet ProdN_1 \bullet I_1 \bullet ProdN_2 \bullet 1_1 \bullet I_2 \bullet 1_2 \bullet N_2 \bullet ProdN_3 \bullet ProdN_4 \bullet I_4$

Для определения отношений между секциями данных можно показать профиль секций данных в методе. Для нашего примера профиль секций данных приведен в табл. 14.1.

Таблица 14.1. Профиль секций данных для метода SumAndProduct

SumN	ProdN	Оператор
		procedure SumAndProduct
1	1	(Niinteger;
1	1	varSumN, ProdNiinteger)
		var
1	1	l:integer;
		begin
2		SumN:=0
	2	ProdN:=1
3	3	for l:=1 to N do begin
3		SumN:=SumN+l
	3	ProdN:=ProdN*l
		end
		end;

Видно, что в столбце переменной для каждой секции указывается количество лексем из *i*-й строки метода, которые включаются в секцию.

Еще одно базовое понятие методики — *секционированная абстракция*. *Секционированная абстракция* — это объединение всех секций данных метода. Например, секционированная абстракция метода SumAndProduct имеет вид

SA(SumAndProduct) = {$N_1 \cdot SumN_1 \cdot I_1 \cdot SumN_2 \cdot 0_1 \cdot I_2 \cdot I_2 \cdot N_2 \cdot SumN_3 \cdot SumN_4 \cdot I_3$,
$N_1 \cdot ProdN_1 \cdot I_1 \cdot ProdN_2 \cdot I_1 \cdot I_2 \cdot I_2 \cdot N_2 \cdot ProdN_3 \cdot ProdN_4 \cdot I_4$}.

Введем главные определения.

Секционированной абстракцией класса (Class Slice Abstraction) CSA(C) называют объединение секций всех экземплярных переменных класса. Полный набор секций составляют путем обработки всех методов класса.

Склеенными лексемами называют те лексемы данных, которые являются элементами более чем одной секции данных.

Сильно склеенными лексемами называют те склеенные лексемы, которые являются элементами всех секций данных.

Сильная связность по данным (StrongData Cohesion) — это метрика, основанная на количестве лексем данных, входящих во все секции данных для класса. Иначе говоря, сильная связность по данным учитывает количество сильно склеенных лексем в классе *C*, она вычисляется по формуле:

$$\mathrm{SDC}(C) = \frac{|SG(\mathrm{CSA}(C))|}{|\text{лексемы}(C)|},$$

где *SG*(CSA(*C*)) — объединение сильно склеенных лексем каждого из методов класса *C*, лексемы*(C)* — множество всех лексем данных класса *C*.

Таким образом, класс без сильно склеенных лексем имеет нулевую сильную связность по

данным.

Слабая связность по данным (Weak Data Cohesion) — метрика, которая оценивает связность, базируясь на склеенных лексемах. Склеенные лексемы не требуют связывания всех секций данных, поэтому данная метрика определяет более слабый тип связности. Слабая связность по данным вычисляется по формуле:

$$\mathrm{WDC}(C) = \frac{|G(\mathrm{CSA}(C))|}{|\text{лексемы}(C)|},$$

где G(CSA(C)) — объединение склеенных лексем каждого из методов класса. Класс без склеенных лексем не имеет слабой связности по данным. Наиболее точной метрикой связности между секциями данных является *клейкость данных (Data Adhesiveness).* Клейкость данных определяется как отношение суммы из количеств секций, содержащих каждую склеенную лексему, к произведению количества лексем данных в классе на количество секций данных. Метрика вычисляется по формуле:

$$\mathrm{DA}(C) = \frac{\sum d \in G(\mathrm{CSA}(C)"d \in \text{Секции}}{|\text{лексемы}(C)| \times |\mathrm{CSA}(C)|}.$$

Приведем пример. Применим метрики к классу, профиль секций которого показан в табл. 14.2.

Таблица 14.2. Профиль секций данных для класса Stack

array top size			**Класс Stack**
			class Stack {int *array, top, size;
			public:
			Stack (int *s*) {
2 2			size=s;
2 2			array=new int [size];
2			top=0;}
			int IsEmpty () {
2			return top==0};
			int Size (){
2			return size};
			intVtop(){
3 3			return array [top-1]; }
			void Push (int item) {
2	2	2	if (top= =size)
			printf ("Empty stack. \n");
			else
3	3	3	array [top++]=item;}
			int Pop () {
1			if (IsEmpty ())

```
                    printf ("Full stack. \n");
                    else
 1                  --top;}
                    };
```

Очевидно, что CSA(Stack) включает три секции с 19 лексемами, имеет 5 сильно склеенных лексем и 12 склеенных лексем.

Расчеты по рассмотренным метрикам дают следующие значения:

SDC(CSA(Stack)) = 5/19 = 0,26

WDC(CSA(Stack)) = 12/19 = 0,63

DA(CSA(Stack)) =(7*2 + 5*3)/(19*3) = 0,51

Метрики связности по методам

Д. Биемен и Б. Кенг предложили метрики связности класса, которые основаны на прямых и косвенных соединениях между парами методов. Если существуют общие экземплярные переменные (одна или несколько), используемые в паре методов, то говорят, что эти методы соединены прямо. Пара методов может быть соединена косвенно, через другие прямо соединенные методы.

На рис. 14.2 представлены отношения между элементами класса Stack. Прямоугольниками обозначены методы класса, а овалами — экземплярные переменные. Связи показывают отношения использования между методами и переменными.

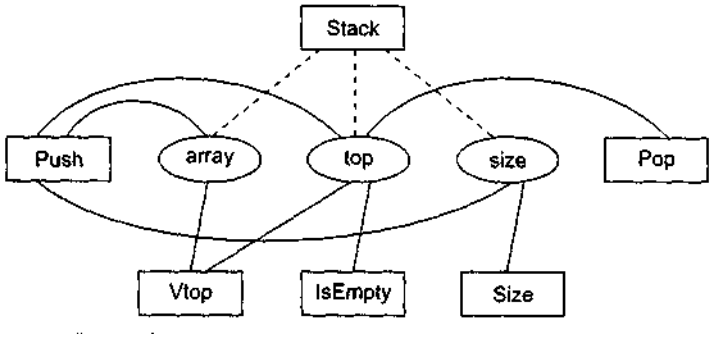

Рис. 14. 2. Отношения между элементами класса Stack

Из рисунка видно, что экземплярная переменная top используется методами Stack, Push, Pop, Vtop и IsEmpty. Таким образом, все эти методы попарно прямо соединены. Напротив, методы Size и Pop соединены косвенно: Size соединен прямо с Push, который, в свою очередь, прямо соединен с Pop. Метод Stack является конструктором класса, то есть функцией инициализации. Обычно конструктору доступны все экземплярные переменные класса, он использует эти переменные совместно со всеми другими методами. Следовательно, конструкторы создают соединения и между такими методами, которые никак не связаны друг с другом. Поэтому ни конструкторы, ни деструкторы здесь не учитываются. Связи между конструктором и экземплярными переменными на рис. 14.2 показаны пунктирными линиями.

Для формализации модели вводятся понятия абстрактного метода и абстрактного класса.

Абстрактный метод AM(M) — это представление реального метода *M* в виде множества *экземплярных* переменных, которые прямо или косвенно используются методом.

Экземплярная переменная прямо используется методом *M,* если она появляется в методе как лексема данных. Экземплярная переменная может быть определена в том же классе, что и *M,* или же в родительском классе этого класса. Множество экземплярных переменных, прямо используемых методом *M,* обозначим как *DU(M).*

Экземплярная переменная косвенно используется методом *M,* если: 1) экземплярная переменная прямо используется другим методом *M',* который вызывается (прямо или косвенно) из метода *M;* 2) экземплярная переменная, прямо используемая методом *M',* находится в том же объекте, что и *M.*

Множество экземплярных переменных, косвенно используемых методом *M,* обозначим как *IU(M).*

Количественно абстрактный метод формируется по выражению:

AM *(M)* = DU *(M)* ∪ IU *(M).*

Абстрактный класс AC(C) — это представление реального класса *C* в виде совокупности абстрактных методов, причем каждый абстрактный метод соответствует видимому методу класса *C.* Количественно абстрактный класс формируется по выражению:

AC *(C)* = [[AM *(M)* | *M* ∈ V *(C)*]],

где *V(C)* — множество всех видимых методов в классе *C* и в классах — предках для *C.*

Отметим, что AM-представления различных методов могут совпадать, поэтому в AC могут быть дублированные элементы. В силу этого AC записывается в форме мультимножества (двойные квадратные скобки рассматриваются как его обозначение).

Локальный абстрактный класс LAC(C) — это совокупность абстрактных методов, где каждый абстрактный метод соответствует видимому методу, определенному только внутри класса *C.* Количественно абстрактный класс формируется по выражению:

LAC*(C)*=[[AM*(M)*|*M* ∈ LV*(C)*]],

где LV*(C)* — множество всех видимых методов, определенных в классе *C.*

Абстрактный класс для стека, приведенного в табл. 14.2, имеет вид:

AC (Stack) = [[{top}, {size}, {array, top}, {array, top, size}, {pop}]].

Поскольку класс Stack не имеет суперкласса, то справедливо:

AC (Stack) = LAC (Stack)

Пусть NP*(C)* — общее количество пар абстрактных методов в AC*(C).* NP определяет максимально возможное количество прямых или косвенных соединений в классе. Если в классе *C* имеются *N* методов, тогда NP*(C) = N*(N*- l)/2. Обозначим:

- NDC*(C)* — количество прямых соединений AC(Q;
- NIC*(C)* — количество косвенных соединений в AC(C).

Тогда метрики связности класса можно представить в следующем виде:

- *сильная связность класса (Tight Class Cohesion (TCC))* определяется относительным количеством прямо соединенных методов:

TCC *(C)* = NDC *(C)* / NP (C);

- *слабая связность класса (Loose Class Cohesion (LCC))* определяется относительным количеством прямо или косвенно соединенных методов:

LCC *(C)* = (NDC *(C)* + NIC *(C)*) / NP *(C)*.

Очевидно, что всегда справедливо следующее неравенство:

LCC*(C)*>=TCC*(C)*.

Для класса Stack метрики связности имеют следующие значения:

TCC(Stack)=7/10=0,7

LCC(Stack)=10/10=l

Метрика TCC показывает, что 70% видимых методов класса Stack соединены прямо, а метрика LCC показывает, что все видимые методы класса Stack соединены прямо или косвенно.

Метрики TCC и LCC индицируют степень связанности между видимыми методами класса. Видимые методы либо определены в классе, либо унаследованы им. Конечно, очень полезны метрики связности для видимых методов, которые определены только внутри класса — ведь здесь исключается влияние связности суперкласса. Очевидно, что метрики локальной связности класса определяются на основе локального абстрактного класса. Отметим, что для локальной связности экземплярные переменные и вызываемые методы могут включать унаследованные переменные.

Сцепление объектов

В классическом методе Л. Констентайна и Э. Йордана определены шесть типов сцепления, которые ориентированы на процедурное проектирование.

Принципиальное преимущество объектно-ориентированного проектирования в том, что природа объектов приводит к созданию слабо сцепленных систем. Фундаментальное свойство объектно-ориентированного проектирования заключается в скрытости содержания объекта. Как правило, содержание объекта невидимо внешним элементам. Степень автономности объекта достаточно высока. Любой объект может быть замещен другим объектом с таким же интерфейсом.

Тем не менее наследование в объектно-ориентированных системах приводит к другой форме сцепления. Объекты, которые наследуют свойства и операции, сцеплены с их суперклассами. Изменения в суперклассах должны проводиться осторожно, так как эти изменения распространяются во все классы, которые наследуют их характеристики.

Таким образом, сами по себе объектно-ориентированные механизмы не гарантируют минимального сцепления. Конечно, классы — мощное средство абстракции данных. Их введение уменьшило поток данных между модулями и, следовательно, снизило общее сцепление внутри системы. Однако количество типов зависимостей между модулями выросло. Появились отношения наследования, делегирования, реализации и т. д. Более разнообразным стал состав модулей в системе (классы, объекты, свободные функции и процедуры, пакеты). Отсюда вывод: необходимость измерения и регулирования сцепления в объектно-ориентированных системах обострилась.

Рассмотрим объектно-ориентированные метрики сцепления, предложенные М. Хитцем и Б. Монтазери.

Зависимость изменения между классами

Зависимость изменения между классами CDBC (Change Dependency Between Classes) определяет потенциальный объем изменений, необходимых после модификации класса-сервера SC (server class) на этапе сопровождения. До тех пор, пока реальное количество необходимых изменений класса-клиента CC (client class) неизвестно, CDBC указывает количество методов, на которые влияет изменение SC.

CDBC зависит от:

- области видимости изменяемого класса-сервера внутри класса-клиента (определяется типом отношения между CS и CC);
- вида доступа CC к CS (интерфейсный доступ или доступ реализации).

Возможные типы отношений приведены в табл. 14.3, где *n* — количество методов класса CC, ъ — количество методов CC, потенциально затрагиваемых изменением.

Таблица 14.3. Вклад отношений между клиентом и сервером в зависимость изменения

Тип отношения	ъ
SC не используется классом CC	0
SC — класс экземплярной переменной в классе CC	n
Локальные переменные типа SC используются внутри /-методов класса CC	j
SC является суперклассом CC	n
SC является типом параметра для/-методов класса CC	j
CC имеет доступ к глобальной переменной класса SC	n

Конечно, здесь предполагается, что те элементы класса-сервера SC, которые доступны классу-клиенту CC, являются предметом изменений. Авторы исходят из следующей точки зрения: если класс SC является «зрелой» абстракцией, то предполагается, что его интерфейс более стабилен, чем его реализация. Таким образом, многие изменения в реализации SC могут выполняться без влияния на его интерфейс. Поэтому вводится фактор стабильности интерфейса для класса-сервера, он обозначается как к ($0 < k < 1$). Вклад доступа к интерфейсу в зависимость изменения можно учесть умножением на $(1 - k)$.

Метрика для вычисления степени CDBC имеет вид:

$$A = \sum_{\substack{\textit{Доступ i} \\ \text{к реализации}}} \alpha_i + (1-k) \times \sum_{\substack{\textit{Доступ i} \\ \text{к интерфейсу}}} \alpha_i ;$$

CDBC(CC, SC) = min(n, A).

Пути минимизации CDBC:

1) ограничение доступа к интерфейсу класса-сервера;

2) ограничение видимости классов-серверов (спецификаторами доступа public, protected, private).

Локальность данных

Локальность данных LD (Locality of Data) — метрика, отражающая качество абстракции, реализуемой классом. Чем выше локальность данных, тем выше самодостаточность класса. Эта характеристика оказывает сильное влияние на такие внешние характеристики, как повторная используемость и тестируемость класса.

Метрика LD представляется как отношение количества локальных данных в классе к общему количеству данных, используемых этим классом.

Будем использовать терминологию языка C++. Обозначим как $M_i(1 \le i \le n)$ методы класса. В их число не будем включать методы чтения/записи экземплярных переменных. Тогда формулу для вычисления локальности данных можно записать в виде:

$$LD = \frac{\sum_{i=1}^{a} |L_i|}{\sum_{i=1}^{a} |T_i|},$$

где:

- $L_i(1 \le i \le n)$ — множество локальных переменных, к которым имеют доступ методы M_i (прямо или с помощью методов чтения/записи). Такими переменными являются: непубличные экземплярные переменные класса; унаследованные защищенные экземплярные переменные их суперклассов; статические переменные, локально определенные в M_i ;
- $T_i(1 \le i \le n)$ — множество всех переменных, используемых в M_i, кроме динамических локальных переменных, определенных в M_i.

Для обеспечения надежности оценки здесь исключены все вспомогательные переменные, определенные в M_i , — они не играют важной роли в проектировании.

Защищенная экземплярная переменная, которая унаследована классом *C*, является локальной переменной для его экземпляра (и следовательно, является элементом L_i), даже если она не объявлена в классе *C*. Использование такой переменной методами класса не вредит локальности данных, однако нежелательно, если мы заинтересованы уменьшить значение CDBC.

Набор метрик Чидамбера и Кемерера

В 1994 году С. Чидамбер и К. Кемерер (Chidamber и Кетегег) предложили шесть проектных метрик, ориентированных на классы. Класс — фундаментальный элемент объектно-ориентированной (ОО) системы. Поэтому измерения и метрики для отдельного класса, иерархии классов и сотрудничества классов бесценны для программного инженера, который

должен оценить качество проекта.

Набор Чидамбера-Кемерера наиболее часто цитируется в программной индустрии и научных исследованиях. Рассмотрим каждую из метрик набора.

Метрика 1: Взвешенные методы на класс WMC (Weighted Methods Per Class)

Допустим, что в классе C определены n методов со сложностью $c_1...,c_2,..., c_n$. Для оценки сложности может быть выбрана любая метрика сложности (например, цикломатическая сложность). Главное — нормализовать эту метрику так, чтобы номинальная сложность для метода принимала значение 1. В этом случае

$$WMC = \sum_{i=1}^{n} C_i$$

Количество методов и их сложность являются индикатором затрат на реализацию и тестирование классов. Кроме того, чем больше методов, тем сложнее дерево наследования (все подклассы наследуют методы их родителей). С ростом количества методов в классе его применение становится все более специфическим, тем самым ограничивается возможность многократного использования. По этим причинам метрика WMC должна иметь разумно низкое значение.

Очень часто применяют упрощенную версию метрики. При этом полагают C_i= 1, и тогда WMC — количество методов в классе.

Оказывается, что подсчитывать количество методов в классе достаточно сложно. Возможны два противоположных варианта учета.

1. Подсчитываются только методы текущего класса. Унаследованные методы игнорируются. Обоснование — унаследованные методы уже подсчитаны в тех классах, где они определялись. Таким образом, инкрементность класса — лучший показатель его функциональных возможностей, который отражает его право на существование. Наиболее важным источником информации для понимания того, что делает класс, являются его собственные операции. Если класс не может отреагировать на сообщение (например, в нем отсутствует собственный метод), тогда он пошлет сообщение родителю.
2. Подсчитываются методы, определенные в текущем классе, и все унаследованные методы. Этот подход подчеркивает важность пространства состояний в понимании класса (а не инкрементности класса).

Существует ряд промежуточных вариантов. Например, подсчитываются текущие методы и методы, прямо унаследованные от родителей. Аргумент в пользу данного подхода — на поведение дочернего класса наиболее сильно влияет специализация родительских классов.

На практике приемлем любой из описанных вариантов. Главное — не менять вариант учета от проекта к проекту. Только в этом случае обеспечивается корректный сбор метрических данных.

Метрика WMC дает относительную меру сложности класса. Если считать, что все методы

имеют одинаковую сложность, то это будет просто количество методов в классе. Существуют рекомендации по сложности методов. Например, М. Лоренц считает, что средняя длина метода должна ограничиваться 8 строками для Smalltalk и 24 строками для C++ [45]. Вообще, класс, имеющий максимальное количество методов среди классов одного с ним уровня, является наиболее сложным; скорее всего, он специфичен для данного приложения и содержит наибольшее количество ошибок.

Метрика 2: Высота дерева наследования DIT (Depth of Inheritance Tree)

DIT определяется как максимальная длина пути от листа до корня дерева наследования классов. Для показанной на рис. 14.3 иерархии классов метрика DIT равна 3.

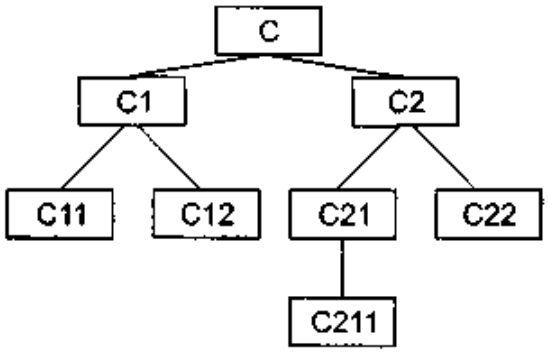

Рис. 14.3. Дерево наследования классов

Соответственно, для отдельного класса DIT, это длина максимального пути от данного класса до корневого класса в иерархии классов.

По мере роста DIT вероятно, что классы нижнего уровня будут наследовать много методов. Это приводит к трудностям в предсказании поведения класса. Высокая иерархия классов (большое значение DIT) приводит к большей сложности проекта, так как означает привлечение большего количества методов и классов.

Вместе с тем, большое значение DIT подразумевает, что многие методы могут использоваться многократно.

Метрика 3: Количество детей NOC (Number of children)

Подклассы, которые непосредственно подчинены суперклассу, называются его детьми. Значение NOC равно количеству детей, то есть количеству непосредственных наследников класса в иерархии классов. На рис. 14.3 класс *C*2 имеет двух детей — подклассы *C*21 и *C*22.

С увеличением NOC возрастает многократность использования, так как наследование — это форма повторного использования.

Однако при возрастании NOC ослабляется абстракция родительского класса. Это означает, что в действительности некоторые из детей уже не являются членами родительского класса и могут быть неправильно использованы.

Кроме того, количество детей характеризует потенциальное влияние класса на проект. По мере роста NOC возрастает количество тестов, необходимых для проверки каждого ребенка.

Метрики DIT и NOC — количественные характеристики формы и размера структуры классов. Хорошо структурированная объектно-ориентированная система чаще бывает организована как лес классов, чем как сверхвысокое дерево. По мнению Г. Буча, следует строить сбалансированные по высоте и ширине структуры наследования: обычно не выше, чем 7 ± 2 уровня, и не шире, чем 7 + 2 ветви [22].

Метрика 4: Сцепление между классами объектов CBO (Coupling between object classes)

CBO — это количество сотрудничеств, предусмотренных для класса, то есть количество классов, с которыми он соединен. Соединение означает, что методы данного класса используют методы или экземплярные переменные другого класса.

Другое определение метрики имеет следующий вид: CBO равно количеству сцеплений класса; сцепление образует вызов метода или свойства в другом классе.

Данная метрика характеризует статическую составляющую внешних связей классов.

С ростом CBO многократность использования класса, вероятно, уменьшается. Очевидно, что чем больше независимость класса, тем легче его повторно использовать в другом приложении.

Высокое значение CBO усложняет модификацию и тестирование, которое следует за выполнением модификации. Понятно, что, чем больше количество сцеплений, тем выше чувствительность всего проекта к изменениям в отдельных его частях. Минимизация межобъектных сцеплений улучшает модульность и содействует инкапсуляции проекта.

CBO для каждого класса должно иметь разумно низкое значение. Это согласуется с рекомендациями по уменьшению сцепления стандартного программного обеспечения.

Метрика 5: Отклик для класса RFC (Response For a Class)

Введем вспомогательное определение. Множество отклика класса RS — это множество методов, которые могут выполняться в ответ на прибытие сообщений в объект этого класса. Формула для определения RS имеет вид

$$RS = \{M\} U_{all\_i} \{R_i\},$$

где $\{R_i\}$ — множество методов, вызываемых методом, $\{M\}$ — множество всех методов в классе.

Метрика RFC равна количеству методов во множестве отклика, то есть равна мощности этого множества:

RFC – card{RS}.

Приведем другое определение метрики: RFC — это количество методов класса плюс количество методов других классов, вызываемых из данного класса.

Метрика RFC является мерой потенциального взаимодействия данного класса с другими

классами, позволяет судить о динамике поведения соответствующего объекта в системе. Данная метрика характеризует динамическую составляющую внешних связей классов.

Если в ответ на сообщение может быть вызвано большое количество методов, то усложняются тестирование и отладка класса, так как от разработчика тестов требуется больший уровень понимания класса, растет длина тестовой последовательности.

С ростом RFC увеличивается сложность класса. Наихудшая величина отклика может использоваться при определении времени тестирования.

Метрика 6: Недостаток связности в методах LCOM (Lack of Cohesion in Methods)

Каждый метод внутри класса обращается к одному или нескольким свойствам (экземплярным переменным). Метрика *LCOM* показывает, насколько методы не связаны друг с другом через свойства (переменные). Если все методы обращаются к одинаковым свойствам, то LCOM = 0.

Введем обозначения:

- НЕ СВЯЗАНЫ — количество пар методов без общих экземплярных переменных;
- СВЯЗАНЫ — количество пар методов с общими экземплярными переменными.
- I_j— набор экземплярных переменных, используемых методом M_j

Очевидно, что

НЕ СВЯЗАНЫ = card $\{I_{ij} \mid I_i \cap I_j = 0\}$,

СВЯЗАНЫ = card $\{I_{ij} \mid I_i \cap I_j \neq 0\}$.

Тогда формула для вычисления недостатка связности в методах примет вид

$$LCOM = \begin{cases} \text{НЕ СВЯЗАНЫ - СВЯЗАНЫ, если (НЕСВЯЗАНЫ > СВЯЗАНЫ);} \\ \text{0 в противном случае.} \end{cases}$$

Можно определить метрику по-другому: LCOM — это количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь.

Рассмотрим примеры применения метрики LCOM.

Пример 1: В классе имеются методы: *M1, M2, M3, M4.* Каждый метод работает со своим набором экземплярных переменных:

$I_1=\{a, b\}$; $I_2=\{a, c\}$; $I_3=\{x, y\}$; $I_4=\{m, n\}$.

В этом случае

НЕ СВЯЗАНЫ = card (I_{13}, I_{14}, I_{23}, I_{24}, I_{34}) = 5; СВЯЗАНЫ = card (I_{12}) = 1.

LCOM = 5-1=4.

Пример 2: В классе используются методы: *M1, M2, M3.* Для каждого метода задан свой набор экземплярных переменных:

$I_1 = \{a,b\}$;$I_2=\{a,c\}$;$I_3=\{x,y\}$,

НЕ СВЯЗАНЫ = card (I_{13}, I_{23}) = 2; СВЯЗАНЫ = card (I_{12}) = 1,

LCOM = 2- 1 = 1.

Связность методов внутри класса должна быть высокой, так как это содействует инкапсуляции. Если LCOM имеет высокое значение, то методы слабо связаны друг с другом

через свойства. Это увеличивает сложность, в связи с чем возрастает вероятность ошибок при проектировании.

Высокие значения LCOM означают, что класс, вероятно, надо спроектировать лучше (разбиением на два или более отдельных класса). Любое вычисление LCOM помогает определить недостатки в проектировании классов, так как эта метрика характеризует качество упаковки данных и методов в оболочку класса.

Вывод: связность в классе желательно сохранять высокой, то есть следует добиваться низкого значения LCOM.

Набор метрик Чидамбера-Кемерера — одна из пионерских работ по комплексной оценке качества ОО-проектирования. Известны многочисленные предложения по усовершенствованию, развитию данного набора. Рассмотрим некоторые из них.

Недостатком метрики WMC является зависимость от реализации. Приведем пример. Рассмотрим класс, предлагающий операцию интегрирования. Возможны две реализации:

1) несколько простых операций:

Set_interval (min. max)

Set_method (method)

Set_precision (precision)

Set_function_to_integrate (function)

Integrate;

2) одна сложная операция:

Integrate (function, min, max. method, precision)

Для обеспечения независимости от этих реализаций можно определить метрику WMC2:

$$WMC2 = \sum_{i=1}^{n} (\text{Количество параметров } i-\text{го метода}).$$

Для нашего примера WMC2 = 5 и для первой, и для второй реализации. Заметим, для первой реализации WMC = 5, а для второй реализации WMC = 1.

Дополнительно можно определить метрику *Среднее число аргументов метода ANAM* (Average Number of Arguments per Method):

ANAM = WMC2/WMC.

Полезность метрики ANAM объяснить еще легче. Она ориентирована на принятые в ОО-проектировании решения — применять простые операции с малым количеством аргументов, а несложные операции — с многочисленными аргументами.

Еще одно предложение — ввести метрику, симметричную метрике LCOM. В то время как формула метрики LCOM имеет вид:

LCOM = max (0, НЕ СВЯЗАНЫ - СВЯЗАНЫ),

симметричная ей метрика *Нормализованная NLCOM* вычисляется по формуле:

NLCOM = СВЯЗАНЫ/(НЕ СВЯЗАНЫ + СВЯЗАНЫ).

Диапазон значений этой метрики: $0 \le \text{NLCOM} \le 1$, причем чем ближе NLCOM к 1, тем выше связанность класса.

В наборе Чидамбера-Кемерера отсутствует метрика для прямого измерения

информационной закрытости класса. В силу этого была предложена метрика *Поведенческая закрытость информации BIH* (Behaviourial Information Hiding):

BIH - (WEOC/WIEOC),

где WEOC — *взвешенные внешние операции на класс* (фактически это WMC);

WIEOC — *взвешенные внутренние и внешние операции на класс.*

WIEOC вычисляется так же, как и WMC, но учитывает полный набор операций, реализуемых классом. Если BIH = 1, класс показывает другим классам все свои возможности. Чем меньше BIM, тем меньше видимо поведение класса. BIH может рассматриваться и как мера сложности. Сложные классы, вероятно, будут иметь малые значения BIH, а простые классы — значения, близкие к 1. Если класс с высокой WMC имеет значение BIH, близкое к 1, следует выяснить, почему он настолько видим извне.

Использование метрик Чидамбера-Кемерера

Поскольку основу логического представления ПО образует структура классов, для оценки ее качества удобно использовать метрики Чидамбера-Кемерера. Пример расчета метрик для структуры, показанной на рис. 14.4, представлен в табл. 14.4.

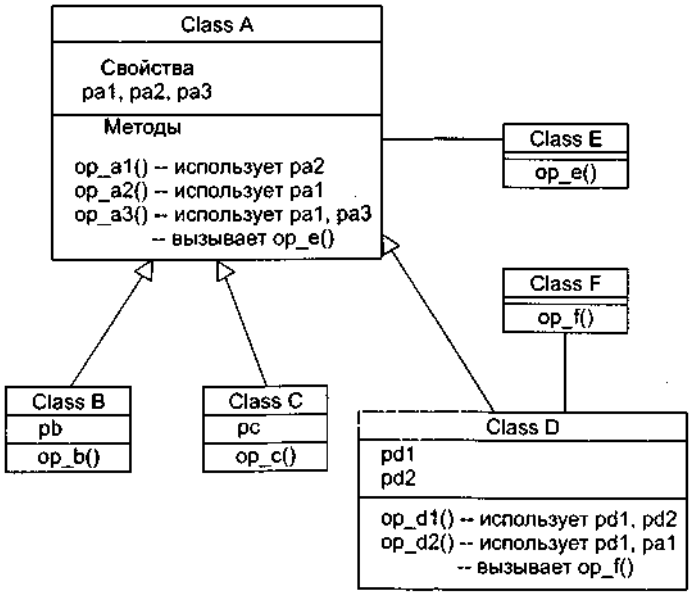

Рис. 14.4. Структура классов для расчета метрик Чидамбера-Кемерера

Прокомментируем результаты расчета. Класс Class A имеет три метода (op_al(), op_a2(), op_a3()), трех детей (Class B, Class C, Class D) и является корневым классом. Поэтому метрики WMC, NOC и DIT имеют, соответственно, значения 3, 3 и 0.

Метрика CBO для класса Class A равна 1, так как он использует один метод из другого класса (метод op_e() из класса Class E, он вызывается из метода op_a3()). Метрика RFC для класса Class A равна 4, так как в ответ на прибытие в этот класс сообщений возможно выполнение четырех методов (три объявлены в этом классе, а четвертый метод op_e() вызывается из op_a3()).

Таблица 14.4. Пример расчета метрик Чидамбера-Кемерера

Имя класса	WMC	DIT	NOC	CBO	RFC	LCOM
Class A	3	0	3	1	4	1
Class B	1	1	0	0	1	0
Class C	1	1	0	0	1	0
Class D	2	1	0	2	3	0

Для вычисления метрики LCOM надо определить количество пар методов класса. Оно рассчитывается по формуле

$C_m^2 = m!/(2(m-2)!)$,

где m — количество методов класса.

Поскольку в классе три метода, возможны три пары: op_al( )&op_a2(), op_al( )&op_a3() и op_a2( )&op_a3(). Первая и вторая пары не имеют общих свойств, третья пара имеет общее свойство (pal). Таким образом, количество несвязанных пар равно 2, количество связанных пар равно 1, и LCOM = 2-1 = 1.

Отметим также, что для класса Class D метрика CBO равна 2, так как здесь используются свойство pal и метод op_f() из других классов. Метрика LCOM в этом классе равна 0, поскольку методы op_dl() и op_d2() связаны по свойству pdl, а отрицательное значение запрещено.