Лекция 8. Метрики объектно-ориентированных программных систем. Продолжение.

Метрики Лоренца и Кидда

Коллекция метрик Лоренца и Кидда — результат практического, промышленного подхода к оценке ОО-проектов.

Метрики, ориентированные на классы

М. Лоренц и Д. Кидд подразделяют метрики, ориентированные на классы, на четыре категории: метрики размера, метрики наследования, внутренние и внешние метрики.

Размерно-ориентированные метрики основаны на подсчете свойств и операций для отдельных классов, а также их средних значений для всей ОО-системы. Метрики наследования акцентируют внимание на способе повторного использования операций в иерархии классов. Внутренние метрики классов рассматривают вопросы связности и кодирования. Внешние метрики исследуют сцепление и повторное использование.

Метрика 1: Размер класса CS (Class Size)

Общий размер класса определяется с помощью следующих измерений:

- общее количество операций (вместе с приватными и наследуемыми экземплярными операциями), которые инкапсулируются внутри класса;
- количество свойств (вместе с приватными и наследуемыми экземплярными свойствами), которые инкапсулируются классом.

Метрика WMC Чидамбера и Кемерера также является взвешенной метрикой размера класса.

Большие значения CS указывают, что класс имеет слишком много обязанностей. Они уменьшают возможность повторного использования класса, усложняют его реализацию и тестирование.

При определении размера класса унаследованным (публичным) операциям и свойствам придают больший удельный вес. Причина — приватные операции и свойства обеспечивают специализацию и более локализованы в проекте.

Могут вычисляться средние количества свойств и операций класса. Чем меньше среднее значение размера, тем больше вероятность повторного использования класса.

Рекомендуемое значение $CS \leq 20$ методов.

Метрика 2: 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 обычно указывают на проблемы проектирования. Ясно, что подкласс должен расширять операции суперкласса. Расширение проявляется в виде новых имен операций. Если же NOO велико, то разработчик нарушает абстракцию суперкласса. Это ослабляет иерархию классов, усложняет тестирование и модификацию программного обеспечения.

Рекомендуемое значение NOO $\leq$ 3 методов.

Метрика 3: Количество операций, добавленных подклассом, NOA (Number of Operations Added by a Subclass)

Подклассы специализируются добавлением приватных операций и свойств. С ростом NOA подкласс удаляется от абстракции суперкласса. Обычно при увеличении высоты иерархии классов (увеличении DIT) должно уменьшаться значение NOA на нижних уровнях иерархии.

Для рекомендуемых значений CS = 20 и DIT = 6 рекомендуемое значение NOA $\leq$ 4 методов (для класса-листа).

Метрика 4: Индекс специализации SI (Specialization Index)

Обеспечивает грубую оценку степени специализации каждого подкласса. Специализация достигается добавлением, удалением или переопределением операций:

SI = (NOO x уровень) /$M_{общ}$,

где *уровень* — номер уровня в иерархии, на котором находится подкласс, $M_{общ}$ — общее количество методов класса.

Пример расчета индексов специализации приведен на рис. 14.5.

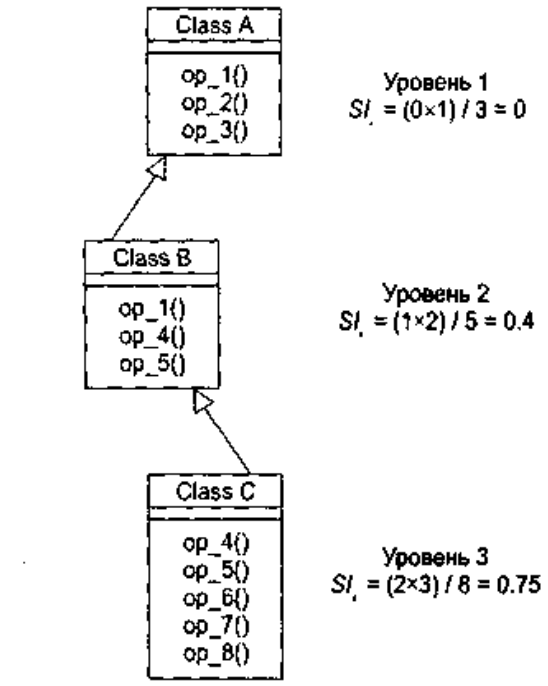

Рис. 14_05

Рис. 14.5. Расчет индексов специализации классов

Чем выше значение SI, тем больше вероятность того, что в иерархии классов есть классы, нарушающие абстракцию суперкласса.

Рекомендуемое значение SI $\leq 0{,}15$.

Операционно-ориентированные метрики

Эта группа метрик ориентирована на оценку операций в классах. Обычно методы имеют тенденцию быть небольшими как по размеру, так и по логической сложности. Тем не менее реальные характеристики операций могут быть полезны для глубокого понимания системы.

Метрика 5: Средний размер операции OS_{AVG} (Average Operation Size)

В качестве индикатора размера может использоваться количество строк программы, однако LOC-оценки приводят к известным проблемам. Альтернативный вариант — *«количество сообщений, посланных операцией»*.

Рост значения метрики означает, что обязанности размещены в классе не очень удачно. Рекомендуемое значение $OS_{AVG} \leq 9$.

Метрика 6: Сложность операции OC (Operation Complexity

Сложность операции может вычисляться с помощью стандартных метрик сложности, то есть с помощью LOC- или FP-оценок, метрики цикломатической сложности, метрики Холстеда.

М. Лоренц и Д. Кидд предлагают вычислять OC суммированием оценок с весовыми

коэффициентами, приведенными в табл. 14.5.

Таблица 14.5. Весовые коэффициенты для метрики ОС

Параметр	Вес
Вызовы функций API	5,0
Присваивания	0,5
Арифметические операции	2,0
Сообщения с параметрами	3,0
Вложенные выражения	0,5
Параметры	0,3
Простые вызовы	7,0
Временные переменные	0,5
Сообщения без параметров	1,0

Поскольку операция должна быть ограничена конкретной обязанностью, желательно уменьшать ОС.

Рекомендуемое значение ОС $\leq$ 65 (для предложенного суммирования).

Метрика 7: Среднее количество параметров на операцию NP_{AVG} (Average Number of Parameters per operation)

Чем больше параметров у операции, тем сложнее сотрудничество между объектами. Поэтому значение NP_{AVG} должно быть как можно меньшим.

Рекомендуемое значение NP_{AVG} = 0,7.

Метрики для ОО-проектов

Основными задачами менеджера проекта являются планирование, координация, отслеживание работ и управление программным проектом.

Одним из ключевых вопросов планирования является оценка размера программного продукта. Прогноз размера продукта обеспечивают следующие ОО-метрики.

Метрика 8: Количество описаний сценариев NSS (Number of Scenario Scripts)

Это количество прямо пропорционально количеству классов, требуемых для реализации требований, количеству состояний для каждого класса, а также количеству методов, свойств и сотрудничеств. Метрика NSS — эффективный индикатор размера программы.

Рекомендуемое значение NSS — не менее одного сценария на публичный протокол подсистемы, отражающий основные функциональные требования к подсистеме.

Метрика 9: Количество ключевых классов NKC (Number of Key Classes)

Ключевой класс прямо связан с коммерческой проблемной областью, для которой предназначена система. Маловероятно, что ключевой класс может появиться в результате повторного использования существующего класса. Поэтому значение NKC достоверно отражает предстоящий объем разработки. М. Лоренц и Д. Кидд предполагают, что в типовой ОО-системе на долю ключевых классов приходится 20-40% от общего количества классов. Как правило, оставшиеся классы реализуют общую инфраструктуру (GUI, коммуникации, базы данных).

Рекомендуемое значение: если NKC < 0,2 от общего количества классов системы, следует углубить исследование проблемной области (для обнаружения важнейших абстракций, которые нужно реализовать).

Метрика 10: Количество подсистем NSUB (NumberofSUBsystem)

Количество подсистем обеспечивает понимание следующих вопросов: размещение ресурсов, планирование (с акцентом на параллельную разработку), общие затраты на интеграцию.

Рекомендуемое значение: NSUB > 3.

Значения метрик NSS, NKC, NSUB полезно накапливать как результат каждого выполненного ОО-проекта. Так формируется метрический базис фирмы, в который также включаются метрические значения по классами и операциям. Эти исторические данные могут использоваться для вычисления метрик производительности (среднее количество классов на разработчика или среднее количество методов на человеко-месяц). Совместное применение метрик позволяет оценивать затраты, продолжительность, персонал и другие характеристики текущего проекта.

Набор метрик Фернандо Абреу

Набор метрик *MOOD* (Metrics for Object Oriented Design), предложенный Ф. Абреу в 1994 году, — другой пример академического подхода к оценке качества ОО-проектирования [6]. Основными целями MOOD-набора являются:

1) покрытие базовых механизмов объектно-ориентированной парадигмы, таких как инкапсуляция, наследование, полиморфизм, посылка сообщений;
2) формальное определение метрик, позволяющее избежать субъективности измерения;
3) независимость от размера оцениваемого программного продукта;
4) независимость от языка программирования, на котором написан оцениваемый продукт.

Набор MOOD включает в себя следующие метрики:

1) фактор закрытости метода (MHF);
2) фактор закрытости свойства (AHF);

3) фактор наследования метода (MIF);

4) фактор наследования свойства (AIF);

5) фактор полиморфизма (POF);

6) фактор сцепления (COF).

Каждая из этих метрик относится к основному механизму объектно-ориентированной парадигмы: инкапсуляции (MHF и AHF), наследованию (MIF и AIF), полиморфизму (POF) и посылке сообщений (COF). В определениях MOOD не используются специфические конструкции языков программирования.

Метрика 1: Фактор закрытости метода MHF (Method Hiding Factor)

Введем обозначения:

- $M_v(C_i)$ — количество видимых методов в классе C_i (интерфейс класса);
- $M_h(C_i)$ — количество скрытых методов в классе C_i (реализация класса);
- $M_d(C_i) = M_v(C_i) + M_h(C_i)$ — общее количество методов, определенных в классе C, (унаследованные методы не учитываются).

Тогда формула метрики MHF примет вид:

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)},$$

где TC — количество классов в системе.

Если видимость *m-го* метода *i*-го класса из *j*-го класса вычислять по выражению:

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1, \text{if} \begin{cases} j \neq 1 \\ C_j \text{ может вызвать } M_{mi} \end{cases} \\ 0, \text{else} \end{cases},$$

а процентное количество классов, которые видят *m*-й метод *i*-го класса, определять по соотношению:

$$V(M_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1}$$

то формулу метрики MHF можно представить в виде:

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)}.$$

В числителе этой формулы MHF — сумма закрытости всех методов во всех классах.

Закрытость метода — процентное количество классов, из которых данный метод невидим. Знаменатель MHF — общее количество методов, определенных в рассматриваемой системе.

С увеличением MHF уменьшаются плотность дефектов в системе и затраты на их устранение. Обычно разработка класса представляет собой пошаговый процесс, при котором к классу добавляется все больше и больше деталей (скрытых методов). Такая схема разработки способствует возрастанию как значения MHF, так и качества класса.

Метрика 2: Фактор закрытости свойства AHF (Attribute Hiding Factor)

Введем обозначения:

- $A_v(C_i)$ — количество видимых свойств в классе C_i (интерфейс класса);
- $A_h(C_i)$ — количество скрытых свойств в классе C_i (реализация класса);
- $A_d(C_i) = A_v(C_i) + A_h(C_i)$ — общее количество свойств, определенных в классе C_i (унаследованные свойства не учитываются).

Тогда формула метрики *AHF* примет вид:

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)},$$

где TC — количество классов в системе.

Если видимость *m*-го свойства *i*-го класса из *j*-го класса вычислять по выражению:

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1, \text{if} \begin{cases} j \neq 1 \\ C_j \text{ может вызвать } A_{mi} \end{cases} \\ 0, \text{else} \end{cases},$$

а процентное количество классов, которые видят *m*-е свойство *i*-го класса, определять по соотношению:

$$V(A_{mi}) = \frac{\sum_{i=1}^{TC} is\_visible(A_{mi}, C_j)}{TC - 1},$$

то формулу метрики *AHF*можно представить в виде:

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)}.$$

В числителе этой формулы AHF — сумма закрытости всех свойств во всех классах. Закрытость свойства — процентное количество классов, из которых данное свойство невидимо. Знаменатель AHF — общее количество свойств, определенных в рассматриваемой системе.

В идеальном случае все свойства должны быть скрыты и доступны только для методов соответствующего класса (AHF = 100%).

Метрика 3: Фактор наследования метода MIF (Method Inheritance Factor)

Введем обозначения:

- $M_i(C_i)$ — количество унаследованных и не переопределенных методов в классе C_i;
- $M_0(C_i)$ — количество унаследованных и переопределенных методов в классе C_i;
- $M_n(C_i)$ — количество новых (не унаследованных и переопределенных) методов в классе C_i;
- $M_d(C_i) = M_n(C_i) + M_0(C_i)$ — количество методов, определенных в классе C_i;
- $M_a(C_i) = M_d(C_i) + M_i(C_i)$ — общее количество методов, доступных в классе C_i.

Тогда формула метрики MIF примет вид:

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_i(C_i)}.$$

Числителем MIF является сумма унаследованных (и не переопределенных) методов во всех классах рассматриваемой системы. Знаменатель MIF — это общее количество доступных методов (локально определенных и унаследованных) для всех классов.

Значение MIF = 0 указывает, что в системе отсутствует эффективное наследование, например, все унаследованные методы переопределены.

С увеличением MIF уменьшаются плотность дефектов и затраты на исправление ошибок. Очень большие значения MIF (70-80%) приводят к обратному эффекту, но этот факт нуждается в дополнительной экспериментальной проверке. Сформулируем «осторожный» вывод: умеренное использование наследования — подходящее средство для снижения плотности дефектов и затрат на доработку.

Метрика 4: Фактор наследования свойства AIF (Attribute Inheritance Factor)

Введем обозначения:

- $A_i(C_i)$ — количество унаследованных и не переопределенных свойств в классе C_i;
- $A_0(C_i)$ — количество унаследованных и переопределенных свойств в классе C_i;
- $A_n(C_i)$ — количество новых (не унаследованных и переопределенных) свойств в классе C_i;
- $A_d(C_i) = A_n(C_i) + A_0(C_i)$ — количество свойств, определенных в классе C_i;
- $A_a(C_i) = A_d(C_i) + A_i(C_i)$ — общее количество свойств, доступных в классе C_i.

Тогда формула метрики AIF примет вид:

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}.$$

Числителем AIF является сумма унаследованных (и не переопределенных) свойств во всех классах рассматриваемой системы. Знаменатель AIF — это общее количество доступных свойств (локально определенных и унаследованных) для всех классов.

Метрика 5: Фактор полиморфизма POF (Polymorphism Factor)

Введем обозначения:

- $M_0(C_i)$ — количество унаследованных и переопределенных методов в классе C_i;
- $M_n(C_i)$ — количество новых (не унаследованных и переопределенных) методов в классе C_i;
- $DC(C_i)$ — количество потомков класса C_i;
- $M_d(C_i) = M_n(C_i) + M_0(C_i)$ — количество методов, определенных в классе C_i.

Тогда формула метрики POF примет вид:

$$POF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]}.$$

Числитель POF фиксирует реальное количество возможных полиморфных ситуаций. Очевидно, что сообщение, посланное в класс C_i связывается (статически или динамически) с реализацией именуемого метода. Этот метод, в свою очередь, может или представляться несколькими «формами», или переопределяться (в потомках C_i).

Знаменатель POF представляет максимальное количество возможных полиморфных ситуаций для класса C_i. Имеется в виду случай, когда все новые методы, определенные в C_i, переопределяются во всех его потомках.

Умеренное использование полиморфизма уменьшает как плотность дефектов, так и затраты на доработку. Однако при POF > 10% возможен обратный эффект.

Метрика 6: Фактор сцепления COF (Coupling Factor)

В данном наборе сцепление фиксирует наличие между классами отношения «клиент-поставщик» (client-supplier). Отношение «клиент-поставщик» (C_c =>C_s) здесь означает, что класс-клиент содержит но меньшей мере одну не унаследованную ссылку на свойство или метод класса-поставщика.

$$is\_client(C_c, C_s) = \begin{cases} 1, \text{if } C_c => C_s \cap C_c \neq C_s, \\ 0, \text{else}, \end{cases}$$

Если наличие отношения «клиент-поставщик» определять по выражению:

$$COF = \frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC} is\_client(C_i, C_j)]}{TC^2 - TC}.$$

то формула для вычисления метрики COF примет вид:

Знаменатель COF соответствует максимально возможному количеству сцеплений в системе с TC-классами (потенциально каждый класс может быть поставщиком для других классов). Из рассмотрения исключены рефлексивные отношения — когда класс является собственным поставщиком. Числитель COF фиксирует реальное количество сцеплений, не относящихся к наследованию.

С увеличением сцепления классов плотности дефектов и затрат на доработку также возрастают. Сцепления отрицательно влияют на качество ПО, их нужно сводить к минимуму. Практическое применение этой метрики доказывает, что сцепление увеличивает сложность, уменьшает инкапсуляцию и возможности повторного использования, затрудняет понимание и усложняет сопровождение ПО.

Метрики для объектно-ориентированного тестирования

Рассмотрим проектные метрики, которые, по мнению Р. Байндера (Binder), прямо влияют на тестируемость ОО-систем. Р. Байндер сгруппировал эти метрики в три категории, отражающие важнейшие проектные характеристики.

Метрики инкапсуляции

К метрикам инкапсуляции относятся: «Недостаток связности в методах LCOM», «Процент публичных и защищенных PAP (Percent Public and Protected)» и «Публичный доступ к компонентным данным PAD (Public Access to Data members)».

Метрика 1: Недостаток связности в методах LCOM

Чем выше значение LCOM, тем больше состояний надо тестировать, чтобы гарантировать отсутствие побочных эффектов при работе методов.

Метрика 2: Процент публичных и защищенных PAP (Percent Public and Protected)

Публичные свойства наследуются от других классов и поэтому видимы для этих классов.

Защищенные свойства являются специализацией и приватны для определенного подкласса. Эта метрика показывает процент публичных свойств класса. Высокие значения *PAP* увеличивают вероятность побочных эффектов в классах. Тесты должны гарантировать обнаружение побочных эффектов.

Метрика 3: Публичный доступ к компонентным данным PAD (Public Access to Data members)

Метрика показывает количество классов (или методов), которые имеют доступ к свойствам других классов, то есть нарушают их инкапсуляцию. Высокие значения приводят к возникновению побочных эффектов в классах. Тесты должны гарантировать обнаружение таких побочных эффектов.

Метрики наследования

К метрикам наследования относятся «Количество корневых классов NOR (Number Of Root classes)», «Коэффициент объединения по входу FIN», «Количество детей NOC» и «Высота дерева наследования DIT».

Метрика 4: Количество корневых классов NOR (Number Of Root classes)

Эта метрика подсчитывает количество деревьев наследования в проектной модели. Для каждого корневого класса и дерева наследования должен разрабатываться набор тестов. С увеличением NOR возрастают затраты на тестирование.

Метрика 5: Коэффициент объединения по входу FIN

В контексте О-О-смистем FIN фиксирует множественное наследование. Значение FIN > 1 указывает, что класс наследует свои свойства и операции от нескольких корневых классов. Следует избегать FIN > 1 везде, где это возможно.

Метрика 6: Количество детей NOC

Название говорит само за себя. Метрика заимствована из набора Чидамбера-Кемерера.

Метрика 7: Высота дерева наследования DIT

Метрика заимствована из набора Чидамбера-Кемерера. Методы суперкласса должны повторно тестироваться для каждого подкласса.

В дополнение к перечисленным метрикам Р. Байндер выделил метрики сложности класса (это метрики Чидамбера-Кемерера — WMC, CBO, RFC и метрики для подсчета количества

методов), а также метрики полиморфизма.

Метрики полиморфизма

Рассмотрим следующие метрики полиморфизма: «Процентное количество не переопределенных запросов OVR», «Процентное количество динамических запросов DYN», «Скачок класса Bounce-C» и «Скачок системы Bounce-S».

Метрика 8: Процентное количество не переопределенных запросов OVR

Процентное количество от всех запросов в тестируемой системе, которые не приводили к перекрытию модулей. Перекрытие может приводить к непредусмотренному связыванию. Высокое значение OVR увеличивает возможности возникновения ошибок.

Метрика 9: Процентное количество динамических запросов DYN

Процентное количество от всех сообщений в тестируемой системе, чьи приемники определяются в период выполнения. Динамическое связывание может приводить к непредусмотренному связыванию. Высокое значение DYN означает, что для проверки всех вариантов связывания метода потребуется много тестов.

Метрика 10: Скачок класса Bounce-C

Количество скачущих маршрутов, видимых тестируемому классу. Скачущий маршрут — это маршрут, который в ходе динамического связывания пересекает несколько иерархий классов-поставщиков. Скачок может приводить к непредусмотренному связыванию. Высокое значение Bounce-C увеличивает возможности возникновения ошибок.

Метрика 11: Скачок системы Bounce-S

Количество скачущих маршрутов в тестируемой системе. В этой метрике суммируется количество скачущих маршрутов по каждому классу системы. Высокое значение Bounce-S увеличивает возможности возникновения ошибок.