Анализ объектно-ориентированных метрик для проектирования архитектуры программного обеспечения.

Романов В.Ю.

***Аннотация* — В статье приводится обзор и анализ объектно-ориентированных метрик используемых для оценки качества проектирования архитектуры программных систем. Сделан анализ простых метрик для оценки качества проекта системы в целом, метрик для оценки проекта на соответствие зарекомендовавшим себя принципам проектирования, метрик оценивающих качество проектирования внутренней структуры классов (связанности класса) и взаимосвязи классов (сцепления классов), метрик оценивающих качество проектирования внутренней структуры пакетов (связанности пакета) и взаимосвязи пакетов (сцепления пакетов).**

***Ключевые слова* — object oriented metrics, software design rules, class cohesion and coupling, package cohesion and coupling.**

I. Введение

Разработка программных систем с помощью библиотек с исходными кодами получает все более распространение. Вместе с тем такой подход к разработке имеет и ряд недостатков. Большой объем исходных текстов этих библиотек, наличие большого числа библиотек со схожей функциональностью, наличие большого количества версий библиотеки, зачастую сопровождаемых различными группами разработчиков, существенно увеличивает время на изучение таких библиотек. Это делает актуальной задачу обратного проектирования (reverse engineering) для построения и визуализации модели такого программного обеспечения [1, 2]. После решения задачи обратного проектирования важным становится извлечение из модели архитектуры программной системы и ее визуализация [3]. В предыдущих работах автора особо рассматривались способы визуализации архитектуры системы [4] и визуализации результатов измерения качества программной системы с помощью объектно-ориентированных метрик[5].

В данной работе приводится обзор и анализ объектно-ориентированных метрик. Рассматриваются простейшие объектно-ориентированные метрики для анализа проектирования отдельных классов. Затем рассматриваются метрики связанности класса, позволяющие оценить качество проектирования структуры класса. Затем рассматриваются метрики сцепления классов, позволяющие оценить качество проектирования взаимосвязей классов. После этого рассматриваются метрики для оценки связанности и сцепления пакетов. В завершение работы рассматриваются метрики для оценки проекта по ряду хорошо зарекомендовавших себя принципов проектирования.

Владимир Юрьевич Романов, факультет ВМК МГУ им. М.В. Ломоносова, vromanov@cs.msu.su, vladimir.romanov@gmail.com.

II. Простейшие объектно-ориентированные метрики

Простейшие метрики описывают основные аспекты исходного кода программы (например, Глубина дерева наследования, Количество методов класса) или являются комбинацией других простейших метрик (например, Количество взвешенных методов на класс, Индекс специализации). Хотя определения таких метрик просты для понимания, интерпретация их значений сильно зависит от используемого языка программирования, контекста измеряемого элемента программы, принятого процесса разработки. Рассмотрим эти метрики подробнее.

Глубина дерева наследования [8] (DIT-Depth of Inheritance Tree). Глубина класса в иерархии наследования есть максимальная длина от узла класса до корня дерева, измеряемая в предках класса. Чем глубже наследование класса в иерархии, тем большее количество методов он, возможно, наследует. И тем более непредсказуемым является поведение класса.

Для данной метрики отсутствует четкое определение контекста. Большая часть цепочки наследования может быть расположена внутри используемой библиотеки классов, в самом приложении может быть расположена ее меньшая часть. Для языка Java следует различать иерархии наследования, возникающие в результате расширения класса и в результате реализации интерфейса. Следует также различать в иерархии классы стандартной библиотеки Java, классы сторонних библиотек и классы собственно приложения.

Количество потомков [9] (Number of Children – NOC) Количество непосредственных потомков класса. Чем больше таких потомков, тем большого тестирования требует данный класс. Поскольку при вычислении метрики учитывается только непосредственные потомки, то оценить качество иерархии наследования данная метрика не позволяет.

Количество методов (Number of Methods - NOM) [10] Количество методов определенных локально в классе, без учета видимости класса. Учитываются также и

переопределенные методы. Данная метрика хорошо подходит для оценки сложности класса.

Количество взвешенных методов на класс (Weighted Methods Per Class WMC [11]
Значение метрики WMC есть сумма цикломатической сложности методов определенных в этом классе.

Цикломатическая сложность (Cyclomatic Complexity Metric - V(G) [12]
Цикломатическая сложность вычисляется как максимальное число линейно независимых участков кода в методе. Участок кода линейный, если не содержит команд ветвления. Значение метрики вычисляется следующим образом:
V (G) = e - n + p
где n есть количество вершин, e есть количество ребер и p есть количество соединенных компонент (узлов выхода) в графе потока управления.
Эта метрика есть индикатор психологической сложности понимания программного кода

III. МЕТРИКИ ПРОЕКТИРОВАНИЯ КЛАССОВ

Эти метрики используются для измерения соответствия программного кода принципам проектирования.

A. Сцепление классов

Метрики сцепления [13, 14] (coupling) предназначены измерения зависимости различных классов друг от друга.

Сцепление между классами объектов [14] (Coupling Between Object classes - CBO).
Два класса сцеплены друг с другом, если один класс использует другой класс. Например, метод одного класса вызывает метод другого класса, или использует атрибуты другого класса. Сцеплением также считается также наследование одного класса от другого класса, или полиморфный вызов другого класса. Значение этой метрики для класса — количество классов с которым он связан. При вычислении этой метрики используются только классы непосредственно связанные друг с другом. Классы с чрезмерным сцеплением сложнее понимать, изменять и корректировать. Метрику CBO следует отличать от метрики центробежного сцепления (подсчитываются только выходящие зависимости) и метрики центростремительного сцепления (подсчитываются только входящие зависимости).

Коэффициент сцепления [15] (Coupling Factor - CF). Область действия этой метрики — не конкретный класс, а программа в целом. Коэффициент сцепления вычисляется как нормализованное соотношение между количеством клиентских отношений и общим числом возможных клиентских отношений. Клиентское отношение существует, если класс ссылается на метод или атрибут другого класса, за исключением случая, когда класс является потомком другого класса. Таким образом, сцепление из-за наследования исключается, но вызываемые полиморфные методы классов — предков учитываются.

Сцепление посылкой сообщений [16] (Message Passing Coupling - MPC). Значение метрики - количество посылаемых сообщений определенных в классе. Вызов собственных методов класса при этом не учитывается. Интерпретация метрики следующая: количество вызовов методов сделанных из класса позволяет измерить насколько реализация методов данного класса зависит от методов в других классах. Полиморфизм методов при этом не учитывается.

B. Связанность классов

Метрики связанности (cohesion) классов позволяют оценить качество проектирования, анализируя связи внутри класса, а не связи между классами, как это делалось с помощью метрик в предыдущем разделе. Связи внутри класса возникают, в частности, из-за связей между методами класса. Связанность методов в классе желательна, поскольку это способствует инкапсуляции связанных методов в одном классе. Отсутствие связанности методов в классе означает, что возможно данный класс следует перепроектировать, расщепив его на два и более подкласса.
Набор метрик «Отсутствие связанности в методах» [15] (Lack of Cohesion in Methods – LCOM) используется для вычисления связанности кода. Для иллюстрации этих метрик используется граф, приведенный на рисунке 1.

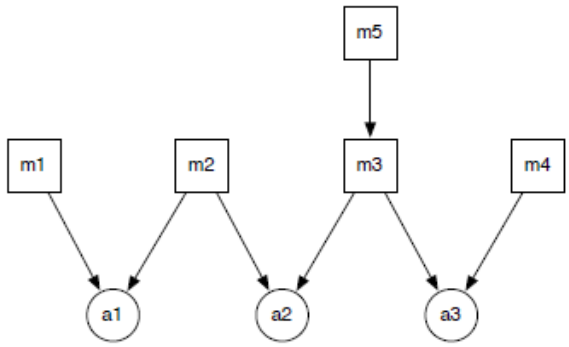

Рис.1. Примерный граф для LCOM метрик.

Метрика LCOM1 представляет количество пар методов в классе, которые не ссылаются на общий атрибут. Для графа, представленного на рисунке 1, LCOM1 = 7. Следующие пары не имеют общего атрибута (m1, m3), (m1, m4), (m2, m4), (m5, m1), (m5, m2), (m5, m3), (m5, m4). Это один из первых способов определения связанности класса. Недостатком метода является использование для вычисления значения метрики только собственных атрибутов и методов класса. Эта метрика учитывает только собственные методы и атрибуты класса.

LCOM2 – для каждой пары методов в классе, если они имеют доступ к различному множеству экземпляров переменных, значение переменной P увеличивается на 1, в противном случае увеличивается значение переменной Q. Если P больше Q, то значение метрики LCOM2 = P-Q. В противном случае значение метрики LCOM2 = 0.
Значение 0 означает, что класс связан. В противном случае класс может быть расщеплен на два и более класса, поскольку их экземпляры переменных принадлежат различным множествам.

Для графа на рисунке 1 значение метрики LCOM2 = 4. Значение переменной P есть сумма всех пар методов, которые ссылаются на общий атрибут. Таким образом, P = 7 (пары (m1, m3), (m1, m4), (m2, m4), (m5, m1), (m5, m2), (m5, m3), (m5, m4)). Значение переменной Q вычисляется из других пар ((m1, m2), (m2, m3), (m3, m4)). Таким образом, Q = 3. Результат P - Q = 4. Эта метрика учитывает только собственные методы и атрибуты класса.

LCOM3 – вычисляет количество соединенных компонент в графе методов. Методы в графе соединены, если методы имеют доступ к тому же атрибуту.
Для графа на рисунке 1 значение метрики LCOM3 = 2. Первая компонента (m1, m2, m3, m4) поскольку эти методы прямо или косвенно соединены через некоторые атрибуты. Вторая компонента (m5) поскольку метод не имеет доступа ни к одному атрибуту. Эта метрика учитывает только собственные методы и атрибуты класса.

LCOM4 – эта метрика вычисляется на основе количества соединенных компонент в графе методов. Методы в графе соединены, если методы имеют доступ к тому же атрибуту. Эта метрика, в отличие от LCOM3, принимает во внимание транзитивный вызов графа.
LCOM4 = 1 означает связанный класс.
LCOM4 >= 2 означает, что этот проблемный класс должен быть разделен на меньшие классы.
LCOM4 = 0 означает, что в классе нет методов.
Для графа, представленного на рисунке 1 значение метрики LCOM4 = 1. Единственная компонента (m1, m2, m3, m4, m5) поскольку эти методы прямо или косвенно связаны с тем же самым набором атрибутов.

LCOM5 – эта метрика вычисляется по формуле:

$$LCOM5 = \frac{NOM - \frac{\sum_{m \in M} NOAcc(m)}{NOA}}{NOM - 1}$$

Где M есть множество методов класса, NOM – количество методов, NOA – количество атрибутов, и NOAcc(m) – количество атрибутов класса доступных методу m. Часто эта метрика обозначается как LCOM*.
Значение метрики LCOM5 нормализовано (изменяется в пределах [0,1]). Благодаря нормализованным значениям возможно сравнение различных версий измеряемых классов.

Плотность связанности класса [15] (Tight Class Cohesion - TTC). Значение этой метрики - нормализованное соотношение между количеством непосредственно связанных через атрибуты методов к общему количеству соединений между методами. Непосредственное соединение между методами существует, если оба метода имеют к атрибуту непосредственно, или через вызов метода, как показано на рисунке 2.

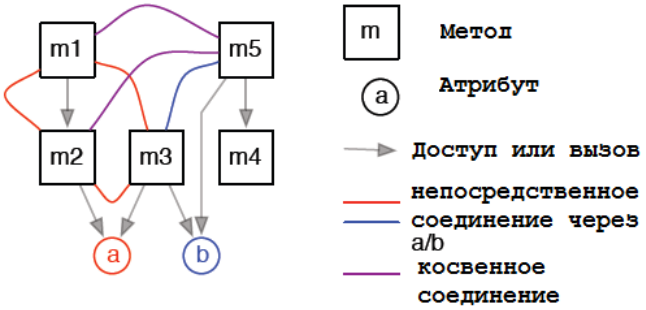

Рис.2. Граф для демонстрации метрики плотности связанности класса.

NP = 0.5 * N / (N-1): максимальное число возможных соединений, где N есть количество видимых методов
NDC: количество непосредственных соединений
TCC = NDC / NP
Метрика TCC принимает значения в пределах [0, 1]. Для этой метрики рассматриваются только видимые методы. Игнорируются конструкторы и деструкторы, а также методы реализующие интерфейсы или обрабатывающие события.
Эта метрика измеряет степень связанности между видимыми методами класса. Большее значение метрики означает большую связанность. Класс со значением метрики меньшим 0.5 несвязан. Конструкторы классов из-за косвенных соединений с атрибутами завышают значение метрики, затрудняя точное измерение метрики.

Потеря сцепления классом [15] (Loose Class Cohesion - LCC)
Значение этой метрики - нормализованное соотношение между количеством непосредственно или косвенно связанных через атрибуты методов к общему количеству соединений между методами. Косвенное соединение между методами существует, если существует последовательность путь от одного метода к другому через последовательность прямых соединений между методами. Это определяется использованием транзитивного замыкания графа, определяемого метрикой TCC.
NP = 0.5 * N / (N - 1): максимальное число возможных соединений, где N есть количество видимых методов
NDC: количество непосредственных соединений
NIC: количество косвенных соединений
LCC = (NDC+NIC) / NP
По определению LCC >= TCC
Метрика TCC принимает значения в пределах [0, 1]. Для этой метрики рассматриваются только видимые методы. Игнорируются конструкторы и деструкторы
Метрика LCC измеряет общую степень связанности между видимыми методами класса. Значение метрики LCC < 0.5 означает несвязанный класс. LCC = 0.8 означает «сильно связанный» класс.
TCC = LCC означает, что класс имеет только прямые связи. TCC = LCC = 1 означает, что все методы класса связаны.

IV. Архитектура пакетов

Оценка пакетов с помощью метрик имеет ряд особенностей. Связанность классов не означает связанность пакетов, в которые классы входят.

Связанность и сцепление пакетов может быть улучшена перемещением классов пакета в другие пакеты или рефакторингом классов, что улучшает связанность и сцепленность классов. В крайнем случае, класс может быть расщеплен на несколько классов для перемещения зависимости в другой класс.

Стоимость сопровождения пакета может сильно изменяться в зависимости от содержания пакета и связей пакета с другими пакетами. По этой причине необходимо иметь метрики для оценки сложности пакета. На связи пакета необходимо обратить особое внимание, поскольку связи между классами пакета становятся связями самого пакета.

Когда рассматривается пакет, необходимо выяснить причину плохого сцепления или связанности пакета. Это может быть неправильно выбранный пакет для класса, каждый класс завязан на самого себя, или неправильное проектирование класса, приведшее к его низкой связанности и сильного сцепления с другим пакетом.

В первом случае структура пакета может быть улучшена перемещением классов. Во втором случае для таких классов необходимо выполнить рефакторинг. Для принятия таких решений необходимы метрики оценки пакета. Начнем с множества простейших метрик характеризующих пакеты.

A. Простейшие метрики пакета

Для иллюстрации простейших метрик пакета на рисунке 3 приводятся изображения четырех пакетов.

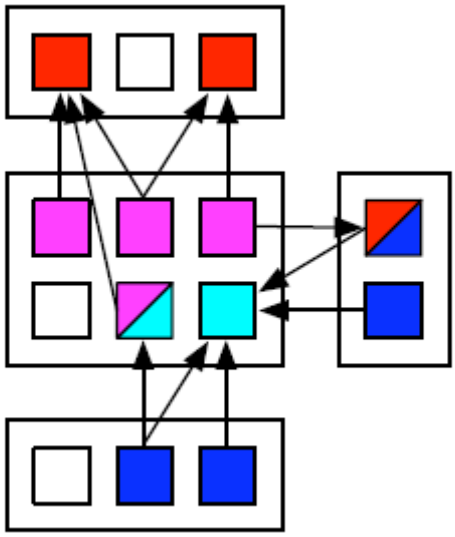

Рис.3. Граф для демонстрации простейших метрик пакетов применимых к центральному пакету на рисунке.

Количество классов в пакете [16] (Number of Classes in Package – NCP). NCP = 6 для центрального пакета на рисунке 3.

Количество экспортируемых классов [16] (Out-port Classes – OutC) – количество классов в пакете, которые зависят от классов из других пакетов. Следует заметить, что это не та же метрика, что и метрика центробежное сцепление. OutC = 4 для центрального пакета на рисунке 3. На рисунке такие классы показаны фиолетовым цветом.

Количество импортируемых классов [16] (In-port Classes – InC) – Количество классов в пакете, от которых зависят классы в других пакетах. Следует заметить, что это не та же метрика, что и метрика центростремительное сцепление. .InC = 2 для центрального пакета на рисунке 3. На рисунке такие классы показаны голубым цветом.

Скрытые классы [16] (Hidden Classes - HC).
Количество классов, которые не имеют связей с классами из других пакетов. HC = 1 для центрального пакета на рисунке 3.

Количество классов - провайдеров системы классов [16] (System Providers Classes – SPC). Количество классов в системе, от которых зависят классы пакета. Следует заметить, что это не та же метрика, что и метрика центробежное сцепление. SPC = 3 для центрального пакета на рисунке 3. На рисунке такие классы показаны красным цветом.

Количество классов – клиентов пакета [16] (System Client Classes – SCC). Количество классов в пакете, от которых зависят классы в других пакетах. Следует заметить, что это не та же метрика, что и метрика центростремительное сцепление. SCC = 4 для центрального пакета на рисунке 3. На этом рисунке такие классы показаны синим цветом.

B. Принципы проектирования и их метрики

Для правильного проектирования архитектуры пакетов хорошо зарекомендовали себя следующие принципы:

Принцип стабильных зависимостей [17] (Stable Dependencies Principle - SDP). Зависимость должна быть направлена в направлении стабильных классов и пакетов. Стабильность связана не только с размером пакета и его сложностью, но и с количеством пакетов от которых он зависит. Пакет с большим числом входящих отношений зависимости из других пакетов, должен быть стабилен, поскольку ответственен за эти пакеты. Пакеты без каких либо входящих зависимостей рассматриваются как независимые и нестабильные.

Принцип стабильных абстракций [17] (Stable Abstractions Principle – SAP). Стабильные пакеты должны быть абстрактными пакетами. Для увеличения гибкости приложений нестабильные пакеты должны быть легко изменяемыми. Стабильные пакеты должны быть легко расширяемыми, и, следовательно, должны быть насколько возможно абстрактными.

Для оценки соответствия проекта указанным принципам был разработан ряд метрик [14]. Вычисление значений этих метрик иллюстрируется на примере графа показанного на рисунке 4.

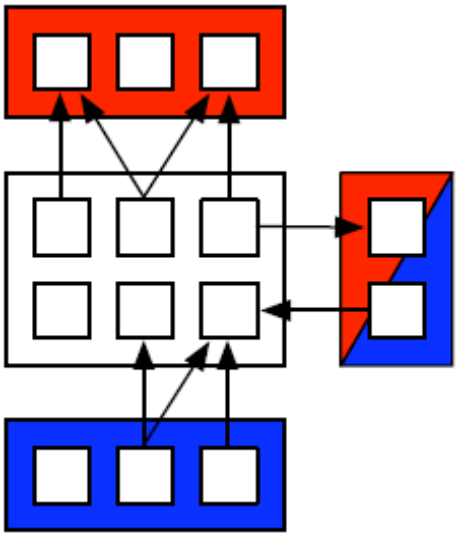

Рис.5. Граф для демонстрации метрик оценки принципам проектирования пакетов применимых к центральному пакету на рисунке

Центробежное сцепление [14] (Efferent Coupling – Ce). Центробежное сцепление модуля – количество модулей от которых он зависит. На рисунке 4 это показывается выходящими зависимостями. На рисунке 4 значение метрики Ce = 2 (количество красных квадратов).

Центростремительное сцепление [14] (Afferent Coupling – Ca). Центростремительное сцепление модуля – количество модулей, которые зависят от этого модуля. На рисунке 4 это показывается входящими зависимостями. На рисунке 4 значение метрики Ca = 2 (количество синих квадратов).

Абстрактность [14] (Abstractness - A). Соотношение между количеством абстрактных классов и общим количеством классов пакета. Значение метрики изменяется в пределах [0..1]. Значение 0 означает полностью конкретный пакет. Значение 1 – полностью абстрактный пакет. Эта метрика должна анализироваться вместе с другими метриками.

Нестабильность [14] (Instability - I).
I = Ce(P) / (Ce(P)+Ca(P)). Значения меняются в пределах [0; 1]. Значение 0 означает максимальную стабильность пакета, поскольку нет зависимостей от других пакетов.
Значение 1 означает нестабильность. Эта метрика используется для оценки соответствия принципу стабильных зависимостей. Пакет должен зависеть только от пакетов, которые более стабильны, чем он сам. Эта метрика дает грубую оценку соответствия принципу SDP. Лучший способ понять метрику нестабильность как пару Ответственный / Независимый. Стабильный пакет со значением I=0 независим и должен быть ответственным, поскольку в него могут входящие зависимости. Нестабильный пакет зависим от других пакетов, и не ответственен за другие пакеты. Понятие стабильности как чувствительности к изменениям, должна быть определена транзитивно: пакет может быть стабильным, если его зависимости также стабильны.

Расстояние [14] (Distance – D)
Значение метрики вычисляется как D = (A+I-1) ./ sqrt(2)
Или, в нормализованном виде DN = A + I - 1.
Пакет должен быть сбалансирован между абстрактностью и нестабильностью. Это правило определяет главную последовательность уравнением
A + I - 1 = 0.
D - это расстояние до главной последовательности.
Эта метрика используется для проверки принципа стабильных абстракций: Стабильные пакеты должны быть также и абстрактными пакетами (A = 1 и I = 0). В тоже время нестабильные пакеты должны быть конкретными. (A = 0 и I = 1).

C. Связанность пакета

Как уже отмечалось, связанность классов в пакете не означает такую же связанность самого пакета. Например, классы пакета могут не иметь связей друг с другом, однако иметь общего предка. И по этой причине оказаться в одном пакете. Внутренней структуры пакета для оценки его связанности недостаточно.
Для оценки связанности пакета в [18] было предложено учитывать контекст пакета. Для оценки пакета была предложена метрика Контекстной связанности пакета (Contextual Package Cohesion - CPC). Эта метрика основана на понятии сходства пар объектов. Чем больше пакетов-клиентов используют схожие классы в пакете, тем больше связанность этого пакета. Иллюстрация этой метрики будет проводиться с помощью рисунка 5.

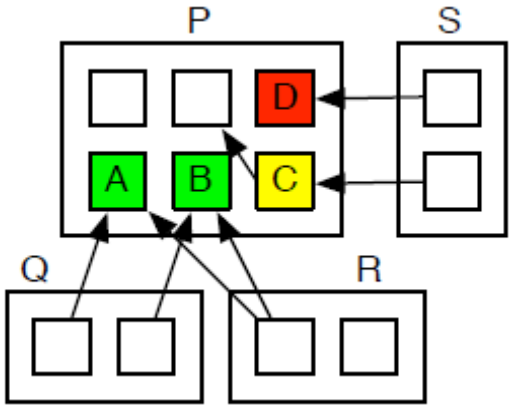

Рис.5. Граф для демонстрации метрик связанности пакетов применимых к центральному пакету на рисунке

Метрика контекстной связанности пакета есть соотношение между количеством пар классов, на которые ссылается общий пакет-клиент, и общим количеством пар классов в интерфейсе пакета. Интерфейс пакета включает все классы, от которых зависят внешние для этого пакета классы. В соответствии с определением метрики CPC связанный пакет тот, у которого классы в интерфейсе зависят от общего множества пакетов. Определяется значение метрики CPC следующим образом:
classes(P) представляет множество классов пакета P.
I = {c | где c принадлежит множеству P^clientClasses(c)-classes(P) != пусто} есть интерфейс пакета P, то есть классы от которых зависят внешние классы.
NP = 0.5 * |I| * (|I| - 1)

Значение метрики есть максимальное число возможных пар сформированное классами интерфейса пакета
f(a, b) = 1
если (clientPackages(a) - P) пересечение
(clientPackages(b) - P) не пустое множество
f(a, b) = 0 в противном случае
CPC =

$$\frac{\sum_{a,b \in I} f(a,b)}{NP}$$

Метрика CPC ориентирована на классы интерфейса пакета и внешних клиентов. Ее значения изменяются в пределах [0..1].
На рисунке 4 показаны классы A, B, C, D интерфейс пакета P. На классы A и B вместе ссылаются пакеты Q и R, делая эту часть пакта связанной. Классы C и D используются только пакетом S и расположены неудачно. Значение метрики CPC для пакета P на рисунке 5 равно 1/6.

Метрика Локальности (Locality) [18]
Эта метрика вычисляется как метрика контекстной связанности пакета с использованием зависимости ссылки. Локальность для класса может быть определена как связанность с классами расположенными не во внешних пакетах, а в том же пакете. На рисунке 5 это связанность с классом C из того же пакета P.

Метрика Удачности (Happiness) [18]
Значение этой метрики позволяет оценить, насколько удачно класс попал в пакет. Она основана на сцеплении класса с классами в других пакетах. Ее значение равно количеству классов-клиентов в этом пакете.
Метрика удачности размещения класса в пакете при рефакторинге пакета должна использоваться совместно с метрикой локальности. Метрика локальности указывает на классы, которые возможно ошибочно расположены в пакете, поскольку имеют различных внешних клиентов. Однако если такой класс также имеет внутренние зависимости (из-за высокого значения меры Удачности), то его перемещение в другой пакет не обязательно.

V. Заключение

В данной работе сделан обзор и анализ объектно-ориентированных метрик. Рассмотрены простейшие объектно-ориентированные метрики для анализа проектирования отдельных классов. Затем рассмотрены метрики связанности класса, позволяющие оценить качество проектирования структуры класса. Далее рассмотрены метрики сцепления классов, позволяющие оценить качество проектирования взаимосвязей классов. После этого рассмотрены метрики для оценки связанности и сцепления пакетов. В завершение работы рассмотрены метрики для оценки проекта по ряду хорошо зарекомендовавших себя принципов проектирования.
Статья является продолжением цикла публикаций по программной инженерии и применению UML, начатой в журнале INJOIT работами [1,2,4,5], а также отраженной в более ранних публикациях [3, 6]. Эта работа относится к числу одного из направлений исследований в Лаборатории ОИТ факультета ВМК МГУ [7].

Библиография

[1] Романов В.Ю. Инструмент обратного проектирования и рефакторинга программного обеспечения написанного на языке Java //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 8. – С. 1-6.
[2] Романов В.Ю. Моделирование свободно-распространяемого программного обеспечения с помощью языка UML //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 7. – С. 11-15.
[3] Романов В.Ю. Моделирование и верификация архитектуры программного обеспечения разработанного на языке Java. Сб. трудов VIII Международной конференции «Современные информационные технологии и ИТ-образование», Москва, 2013, с. 343-348
[4] Романов В. Ю. Визуализация для измерения и рефакторинга программного обеспечения //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 9. – С. 1-10.
[5] Романов В.Ю. Визуализация программных метрик при описании архитектуры программного обеспечения //International Journal of Open Information Technologies. – 2014. – Т. 2. – №. 2. – С. 21-28.
[6] Романов В.Ю. Реализация метамодели языка UML на основе хранилища данных фирмы Google. Сб. трудов VII Международной научно-практической конференции "Современные информационные технологии и ИТ-образование". М.:, 2012. с.605-610.
[7] Намиот Д., Сухомлин В. О проектах лаборатории ОИТ //International Journal of Open Information Technologies. – 2013. – Т. 1. – №. 5. – С. 18-21.
[8] Ewan Tempero, James Noble, and Hayden Melton. How do java programs use inheritance? an empirical study of inheritance in java software. In ECOOP '08: Proceedings of the 22nd European conference on Object-Oriented Programming, pages 667–691, Berlin, Heidelberg, 2008. Springer-Verlag.
[9] Tibor Gyimóthy, Rudolf Ferenc, and Istvá Siket. Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transactions on Software Engineering, 31(10):897–910, 2005.
[10] Mark Lorenz and Jeff Kidd. Object-Oriented Software Metrics: A Practical Guide. Prentice-Hall, 1994.
[11] Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. IEEE Transactions on Software Engineering, 20(6):476–493, June 1994.
[12] T.J. McCabe. A measure of complexity. IEEE Transactions on Software Engineering, 2(4):308–320, December 1976.
[13] W. P. Stevens, G. J. Myers, and L. L. Constantine. Structured design. IBM Systems Journal, 13(2):115–139, 1974.
[14] Robert C. Martin. The tipping point: Stability and instability in oo design, 2005. http://www.ddj.com/architect/184415285.
[15] Lionel C. Briand, John W. Daly, and Jurgen K. Wust. A Unified Framework for Coupling Measurement in Object-Oriented Systems. IEEE Transactions on Software Engineering, 25(1):91–121, 1999.
[16] Hani Abdeen. Visualizing, Assessing and Re-Modularizing Object-Oriented Architectural Elements. PhD thesis, Université of Lille, 2009.
[17] Robert C. Martin. Design principles and design patterns, 2000. www.objectmentor.com.
[18] María Laura Ponisio. Exploiting Client Usage to Manage Program Modularity. PhD thesis, University of Bern, Bern, June 2006.

Object Oriented Metrics Analysis for Software Architecture Design

Romanov V.Y.

***Abstract* — the article provides a review of object oriented metrics used for software architecture quality design. The simple object oriented metrics analysis has been made. The object oriented metrics have been built to assess design rule principles. Class coupling and class cohesion metrics were reviewed. Package coupling and class cohesion metrics were reviewed too.**

***Keywords* — object oriented metrics, software design rules, class cohesion and coupling, package cohesion and coupling.**