УДК 681.007.05

А.В. Липанов

Харьковский национальный университет радиоэлектроники, Харьков

МЕТОД ПОСТРОЕНИЯ ОБОБЩЕННОЙ ОЦЕНКИ КАЧЕСТВА ИСХОДНОГО КОДА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В статье рассматривается новый подход к получению обобщенной оценки качества исходного кода программного обеспечения (ПО) с использованием метрик. В работе приводится анализ и модификация метрик исходного кода ПО для получения их нормированного значения в интервале [0, 1]. Перевод метрик в метрическое пространство позволяет рассматривать их как некоторую группу, в которой есть определенные связи и факторы взаимного влияния метрик друг на друга. Анализ такого пространства метрик позволяет говорить о мерах близости, анализе сходимости или расходимости метрических групп. Это позволяет получить комплексную оценку качества исходного кода на основании группы метрик, а не одной конкретной метрики. Интегральная оценка исходного кода позволяет осуществлять быстрый контроль направления развития исходного кода.

Ключевые слова: *метрики исходного кода, единство, сцепление, нестабильность, относительное единство, ассоциация между классами, абстрактность, нормирование метрик, экспертная система анализа исходного кода.*

Введение

Контроль качества программного обеспечения (ПО) – это одна из важнейших задач в современных процессах разработки ПО. Существуют разные подходы и методы к контролю качества ПО, включая различные подходы к организации процессов разработки ПО. В то же время, при выполнении различных процедур контроля качества больше внимания уделяется различным аспектам, связанным с функциональностью ПО, производительностью, надежностью и безопасностью. При этом вопросы анализа качества исходного кода, а именно его расширяемости, масштабируемости, соответствия парадигмам ООП, возможности повторного использования, понятности, эффективности – зачастую уходят на второй план. Чаще всего это связано с тем, что в условиях дефицита времени разработчики вынуждены сокращать время везде, где это возможно, в том числе, и на анализ исходного кода, поскольку, как правило, это достаточно сложная и трудоемкая задача. Вместе с этим, экономия времени на разработке архитектуры ПО и на последующем контроле исходного кода может привести к значительно большим временным и финансовым потерям в будущем. Осознание этого факта приводит к тому, что уже много лет назад начали развиваться различные методы анализа исходного кода, в том числе и с использованием метрик исходного кода. Наиболее известной и фундаментальной работой в этом направлении является работа [1] Тома ДеМарко. Основным тезисом своей книги Том ДеМарко сделал утверждение: «Мы не можем управлять тем, что мы не можем измерить». В работе [2] авторы описали значительное количество метрик и подходов к их вычислению. Однако в последнее время все больше стало появляться скептицизма в отношении метрик, в частности сам Том ДеМарко в своей работе [3] высказывает мнение о том, что использование метрик не является обязательным, нужным и способным дать значительную пользу в процессе разработки. Данная ситуация объясняется тем, что существующие на данный момент теоретические наработки и программные продукты, использующие их, не дают ответа на несколько важных вопросов, а именно:

1. Как трактовать значения метрик исходного кода?
2. Что делать дальше и как менять исходный код, если есть вычисленные значения метрик?
3. Как оценить в целом качество исходного кода при наличии значений метрик?
4. Как оценить правильность направления, по которому идет разработка исходного кода ?

Очевидно, что когда нет эффективных алгоритмов и инструментов, позволяющих использовать подходы анализа исходного кода с использованием метрик, то их применение становится малоэффективным и зачастую становится простой потерей времени, поскольку разработчик часто действует в определенном смысле вслепую, модифицируя исходный код с определенной степенью случайности, что, конечно же, не всегда дает хорошие результаты.

Под метриками исходного кода понимаются числовые оценки, вычисляемые по определенным правилам с использованием определенных характеристик исходного кода. Далее в статье метрики, используемые при построении обобщенной оценки качества исходного кода, будут рассмотрены подробно.

Для того чтобы сделать аппарат метрик исходного кода пригодным к использованию, необходимо применить ряд новых и обобщающих подходов, которые позволили бы решать задачу комплексно и с предоставлением конкретных рекомендаций разработчику по улучшению исходного кода. Также необходимо, чтобы разработчик имел эффективные инструменты, позволяющие проводить анализ исходного кода и выдавать рекомендации, содержащие указание фрагментов кода, подлежащих исправлению. При таких условиях аппарат метрик станет существенно более пригодным к использованию.

Попытки решить данные задачи в комплексе предпринимались неоднократно, например, в работе [4] авторы рассматривают взаимосвязи качественных характеристик исходного кода ПО, их связь с метриками исходного кода и парадигмами объектно-ориентированного программирования. В своей работе авторы предложили иерархическую модель качества объектно-ориентированного дизайна, которая опирается на метрики исходного кода, при этом вопрос формирования рекомендаций по улучшению исходного кода остается нерешенным. В связи с тем, что на данный момент нет комплексного подхода к анализу качества исходного кода, то в ходе исследований решались такие задачи:

1. Формирование системы нормированных метрик исходного кода.

2. Разработка алгоритма для получения обобщенной оценки качества исходного кода.

Решение этих задач позволит разработать автоматизированную систему анализа исходного кода ПО с возможностью предоставления рекомендаций по его улучшению. Использование такой системы позволит существенно сократить время на анализ исходного кода, даст возможность предоставить рекомендации по улучшению исходного кода, а значит, использование метрик исходного кода приобретет практическое значение.

1. Нормированные метрики исходного кода

На сегодняшний день известно много метрик исходного кода, которые применяются для статистической оценки объема кода, наличия комментариев, сложности и качества исходного кода. В работе изучаются метрики исходного кода, определяющие его качество. Под качеством исходного кода понимается его соответствие лучшим интервалам значений качественных метрик исходного кода. В свою очередь, это будет означать, что исходный код, метрики которого имеют «хорошие» значения, соответствует парадигмам ООП, является эффективным, масштабируемым, понятным, не содержит повторяющихся фрагментов, а также может использоваться повторно.

В работе используются метрики, приведенные в табл. 1. Метрики, приведенные в табл. 1, вычисляются по различным формулам. При этом каждая из этих метрик имеет различные интервалы значений.

Таблица 1

Качественные метрики исходного кода

	Номер уровня (j)	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Номер метрики (i)	**Метрика**	**Пространства имен или пакеты**	**Типы**	**Классы**	**Методы**	**Интерфейсы**	**Структуры**	**Перечисления**
1	Сцепление	+		+		+	+	
2	Центростремительное сцепление	+		+		+	+	
3	Внешнее сцепление	+		+		+	+	
4	Нестабильность	+						
5	Относительное единство	+						
6	Расстояние от главной последовательности	+						
7	Абстрактность	+	+	+		+	+	+
8	Связность LCOM		+	+	+			
9	Связность LCOMHS		+	+	+			
10	Ассоциация между классами		+	+		+	+	+

Для того, чтобы можно было эффективно использовать метрики при построении обобщенной системы анализа исходного кода, необходимо нормировать значения метрик в интервал [0,1]. В работе [5] уже использовался подход нормирования значений метрик, однако там использовалось всего 3 метрики.

В данной работе приводится существенно расширенный перечень нормированных метрик, приведены упрощенные вычислительные формулы для некоторых метрик, а также введен ряд новых метрик.

Далее рассмотрим эти метрики и способы их нормирования в интервал [0,1].

1.1. Сцепление

Данная метрика показывает, насколько сильно сцеплен код, т.е. насколько сильно классы, методы, параметры методов связаны между собой, ссылаются друг на друга. Низкое сцепление говорит о том, что исходный код организован таким образом, что его методы и классы мало обращаются друг к другу, а значит код написан неоптимально, поскольку для решения отдельных задач создавались достаточно автономные методы и классы. Метрика может вычисляться разными способами для разных уровней исходного кода и, следовательно, сцепление исследуется на разных уровнях.

Диапазон значений метрики (0,67–1). Значение сцепления тем больше, чем больше модуль сцеплен и находится в пределах от 0,67 до 1. В зависимости от того, какой конкретно вариант сцепления вычисляется, в знаменателе используются различные значения, указанные ниже. Согласно описания в [6], в общем случае сцепление вычисляется по формуле, имеющей вид:

$$C = 1 - \frac{1}{d_i + 2c_i + d_0 + 2c_0 + g_d + 2g_c + w + r}, \quad (1)$$

где используются следующие обозначения:

– для сцепления данных и управления/контроля: d_i – количество входящих данных; c_i – количество входящих управляющих/контрольных параметров; d_o – количество параметров возвращаемых данных; c_o – количество возвращаемых управляющих параметров;

– для глобального сцепления: g_d – количество глобальных переменных, использованных для данных; g_c – количество глобальных переменных, использованных для управления;

– для связности окружения: w – количество вызываемых модулей; r – количество модулей вызывающих данные.

Могут быть рассмотрены и другие варианты сцепления, в частности, они приведены в [5]. Используя данные варианты параметров, можно получать значения сцепления для анализа на различных уровнях исходного кода. Для вычисления связности исходного кода в целом используется:

$$C = r_t \left(\sum_{i=0}^{n} r_i \right)^{-1}, \quad (2)$$

где r_i – количество ссылок на класс, (здесь ссылка – это поле, локальная переменная, возвращаемый тип или параметр метода); r_t – количество ссылок на класс, участвующий в подсчете метрики; n – количество классов.

1.2. Центростремительное сцепление

Метрика применима к пакетам и пространствам имен. Она показывает количество типов за пределами пакета или пространства имен, которые зависят от типов текущего пакета или пространства имен. Высокое центростремительное сцепление показывает, что рассматриваемое пространство имен или пакет имеют большую важность.

$$C_a = 0,35\frac{R}{PC} + 0,65f(N),$$
$$f(N) = \begin{cases} 1, & R \geq N, \\ \dfrac{R}{N}, & R < N, \end{cases} \quad (3)$$

где PC – количество всех типов в исходном коде; N – количество всех типов в рассматриваемом пакете или пространстве имен; R – количество ссылок на типы рассматриваемого пакета или пространства имен из других пакетов или пространств имен.

Таким образом, значения метрики всегда будут находиться в интервале [0, 1], что обеспечит автоматическое нормирование метрики. Хотя при вычислении этой метрики по формуле согласно ее определению, значение этой метрики находится в интервале (0, n), где n – количество всех типов в исходном коде.

1.3. Внешнее сцепление

Количество типов внутри пакета/пространства имен, которые зависят от типов за пределами пространства имен или пакета. Высокое внешнее сцепление показывает, насколько зависимо рассматриваемое пространство имен или пакет от внешних пространств имен или пакетов:

$$C_e = \frac{R}{N}, \quad (4)$$

где C_e – внешнее сцепление; N – количество типов в пакете/пространстве имен; R – количество типов пакета или пространства имен, которые используют типы вне этого типов пакета или пространства имен. Соображения относительно максимального значения данной метрики являются аналогичными тем, которые приведены для метрики Центростремительного сцепления. Следует отметить, что в случае идеального внешнего сцепления R=N.

1.4. Нестабильность

Данная метрика вычисляется с использованием значений центростремительного и внешнего сцеплений. При I=0 метрика указывает на максимально стабильный класс, пакет или пространство имен, а при I = 1 – максимально нестабильный класс, пакет или пространство имен. Маленькое значение метрики означает, что небольшие изменения в классах, пакетах или пространствах имен, от которых зависит фрагмент исходного кода, приведут к большим изменениям в нем, а также изменения в данном фрагменте кода приведут к изменениям в зависимых классах, которые зависят от данного. Формула для вычисления нестабильности из [5] следующая:

$$I = \frac{C_e}{C_a + C_e}, \quad (5)$$

где C_e – внешнее сцепление, C_a –центростремительное сцепление.

1.5. Недостаточность связности LCOM и LCOM HS (Henderson-Sellers)

Существует несколько методов вычисления недостатка связности. Значения LCOM принадлежат промежутку [0,1], значения LCOM HS (Henderson-Sellers) – промежутку [0, 2]. Значение LCOM HS большее, чем 1, свидетельствует о плохой связности. Формулы для подсчета метрики следующие [4, 5]:

$$LCOM = 1 - \frac{1}{M * F}\sum_{i=0}^{n} MF, \quad (6)$$

$$LCOM\ HS = \frac{1}{M-1}\left(M - \frac{1}{F}\sum_{i=0}^{n} MF\right), \quad (7)$$

где M – количество методов в классе (статических и экземплярных конструкторов, геттеров и сеттеров свойств, методов добавления и удаления событий); F – количество экземплярных полей класса; MF – количество методов класса, имеющих доступ к определенному экземплярному полю, $\sum_{i=0}^{n} MF$ – сумма MF по всем экземплярным полям класса.

Основная идея этой метрики заключается в том, что класс абсолютно связан, если все его методы используют все его поля, т.е. $\sum MF = M \times F$ и приводит к тому, что LCOM = 0 и LCOM HS = 0.

Метрики LCOM и LCOM HS показывают, насколько сильно класс связан, т.е. все ли методы данного класса используют все поля этого класса. В идеальном случае, когда класс абсолютно связан, то все методы класса используют все его поля. В случае абсолютной связанности класса значение LCOM и LCOM HS равно 0, и при ухудшении связности LCOM растет до 1, а LCOM HS до 2, причем, при значениях LCOM HS <1 связность класса все еще остается хорошей.

Учитывая тот факт, что значение данной метрики в 0 является наилучшим и при величине метрики 1 (LCOM HS=2) – наихудшим, то необходимо провести инверсию значения метрики при нормировании ее в общую систему метрик. Инверсия производится путем вычитания из 1 значения метрики LCOM и из 2 значения LCOM HS с последующим нормированием метрики LCOM HS в интервал [0,1]. Таким образом, наилучшие значения метрики будут в том случае, если они близки к 1, а наихудшие – когда они близки к 0.

1.6. Относительное единство

Данная метрика показывает среднее число внутренних отношений для каждого типа в пакете или пространстве имен. Через R обозначим число отношений типа внутри данной сборки, т.е. те типы, которые не ссылаются или не используют типы за пределами данного пакета или пространства имен, а N – это количество всех типов внутри пакета/пространства имен. Вычислительную формулу приведем в более строгий вид по отношению к тому, который предлагается в литературе, а именно из числителя уберем добавление 1, чтобы исключить искажение значений метрики и тогда вычислительная формула метрики будет иметь вид (8):

$$H = \begin{cases} \frac{1}{N}, R = 0, \\ \frac{R}{N}, R > 0, \end{cases} \quad (8)$$

где R – количество типов отношений, являющихся внутренними для данного пакета или пространства имен; N – общее количество типов в пакете или пространстве имен.

Считается, что наилучшие значения данной метрики лежат в интервале (1.5, 4). Для проведения нормирования данного интервала в интервал [0,1] нормируем значение метрики по следующей формуле:

$$f(H, r_{max}, r_{min}, n_{max}, n_{min}) = \\ = n_{min} + (H - r_{min}) \begin{cases} \frac{r_{max} - r_{min}}{n_{max} - n_{min}}, & n_{max} > r_{max}, \\ \frac{n_{max} - n_{min}}{r_{max} - r_{min}}, & r_{max} > n_{max}, \end{cases} \quad (9)$$

где r_{max} – максимальное значение для реального интервала; r_{min} – минимальное значение для реального интервала; n_{max} – максимальное значение для нормированного интервала; n_{min} – минимальное значение для нормированного интервала.

Таким образом, получаем функцию вычисления нормированной метрики относительного единства:

$$H_{norm}(H) = \begin{cases} f(H, 4, R, 0, 0.35), & H \in [4, R] \\ f(H, 0, 1.5, 0.35, 0.7), & H \in [0, 1.5] \\ f(H, 1.5, 4, 0.7, 1), & H \in [1.5, 4] \end{cases} \quad (10)$$

Если после нормирования значение не будет попадать в интервал [0,1] т.е. оно будет больше 1, то мы его считаем равным 1, но ставим отдельную отметку, что интервал превышен. Это отмечается на уровне экспертной системы, реализующей данную метрику.

1.7. Абстрактность

Это отношение числа внутренних абстрактных типов (абстрактных классов и интерфейсов) к количеству внутренних типов (классов и интерфейсов). Интервал значений этой метрики [0,1]. Если значение метрики равно 0, то это – абсолютно конкретный пакет или пространство имен, которое не содержит в себе никаких абстрактных интерфейсов и

классов. Если значение метрики равно 1, то это абсолютно абстрактный пакет или пространство имен:

$$A = \frac{M}{N}, \quad (11)$$

где M – количество абстрактных элементов; N – общее количество элементов.

Уровень абстрактности какого-либо пакета или пространства имен показывает степень использования такого принципа ООП как полиморфизм, что в свою очередь, показывает, насколько тот или иной пакет или пространство имен адаптивно к различного рода изменениям. Если значение данной метрики близко к нулю, то это означает, что внутри пакета или пространства имен вообще нет абстрактных классов, а значит, она не описывает никаких сущностей, которые могут использоваться в системе для реализации различных сложных объектов. Низкий уровень абстракции также указывает на то, что исходный код является слишком конкретизированным и его изменения могут быть сложными.

Данная логика расчета метрики абстрактности применима к типам. В данном случае при расчете используется количество свойств и методов.

1.8. Расстояние от главной последовательности

Для вычисления этой метрики необходимо вычислить значение абстрактности A и нестабильности I по соответствующим формулам для данных метрик. Далее можно вычислять данную метрику, которая представляет собой нормализованный перпендикуляр, определяющий расстояние от рассматриваемого пространства имен/пакета к идеальной линии называемой главной последовательностью. Данная метрика является индикатором сбалансированности пространства имен/пакета между абстрактностью и стабильностью. Идеальное пространство имен или пакет является полностью стабильным и абстрактным, т.е. I= 1, A=1 или же оно абсолютно конкретно и не стабильно I=0, A=0. Указанные значения I и A инвертированы, т.е. I=1-I, A=1-A. Это позволяет использовать метрику в диапазоне [0,1]. Если значение метрики равно 0, то это означает, что она совпадает с главной последовательностью, а если значение метрики 1, то это означает, что сборка далека от главной последовательности насколько это возможно. Таким образом, получаем, что наилучшее значение метрики равно 0, а худшее 1. Для того, чтобы метрику можно было отобразить, в нормированном диапазоне модифицируем формулу вычисления метрики следующим образом:

$$DMS = \begin{cases} I - A, & A \in [0, 0.35),\ I \in [0.35, 1], \\ f(|A - I|, 0.35, 0.7), & A, I \in [0.35, 0.7), \\ \frac{A + I}{2}, & A, I \in [0.7, 1] \text{ или } A, I \in [0, 0.35), \end{cases} \quad (12)$$

где A – абстрактность; I – нестабильность.

Для получения нормированного значения DMS используем формулу (13):

$$DMS_{norm} = (max - DMS)\frac{max - min}{max} + min, \quad (13)$$

где min – нижняя граница интервала; max – верхняя граница интервала.

Поскольку данная метрика является балансом между абстрактностью и нестабильностью, то для достижения лучшего показателя данной метрики необходимо добиваться лучшего баланса показателей для абстрактности и нестабильности.

1.9. Ассоциация между классами, интерфейсами, структурами и перечислениями

Метрика «ассоциация между классами» для конкретного класса или структуры – это количество членов других типов или классов, которые непосредственно использует данный конкретный класс в своих методах. Всегда необходимо стремиться, чтобы метод использовал, в основном, элементы собственного класса.

Данная метрика применима для классов, интерфейсов, структур и перечислений и показывает, как много методов, структур, перечислений и интерфейсов использует данный класс из других классов. При помощи данной метрики можно идентифицировать, насколько сильно данный класс зависит от других классов, и, как следствие, показать необходимость реструктуризации определенной части исходного кода.

Например, если класс использует только методы других классов, то это означает, что класс не описывает никакой новой абстракции, а служит «посредником» для доступа к методам другого класса. Также следует учитывать, что родственные элементы данных, используемые вместе, должны быть организованы в классы. Иными словами, если неоднократно используется один и тот же набор элементов данных, то целесообразно объединить эти данные и выполняемые над ними операции в отдельный класс.

Следует отметить, что данная метрика вычисляется отдельно для классов, интерфейсов, структур и перечислений. В [4] приведены вычислительные формулы этой метрики, однако для реализации ее в экспертной системе и с целью ее упрощения мы предлагаем такую формулу:

$$ABC_X = \sum_{i=0}^{N} i \,|\, (F \notin X), \quad (14)$$

где ABC_X – значение метрики для некоторого класса; i – количество обращений к некоторому методу, структуре, интерфейсу или перечислению F, не принадлежащим данному классу X; N – количество всех классов в системе.

При расчете этой метрики очень важно обращать внимание, на то чтобы при вычислении метрики не были учтены стандартные методы, интерфейсы, структуры и перечисления, доступные в языке программирования, на котором разработано данное ПО или же проводить вычисления отдельно, для учета ассоциации со стандартными методами, интерфейсами и т.д. из языка программирования.

В общем случае для каждого класса будет вычислено 4 значения данной метрики – для методов, для интерфейсов, для структур, для перечислений. В случае, если одно из этих значений будет равно нулю, то это означает, что в изучаемом классе не использованы методы, интерфейсы, структуры или перечисления из других классов.

Расширением данной метрики может быть подсчет отношения количества используемых внешних методов, интерфейсов, структур и перечислений к количеству методов, интерфейсов, структур и перечислений из самого класса X. В этом случае вычисление такой модифицированной метрики выполняется по формуле:

$$ABC_X^Y = \frac{\sum_{j=0}^{M} j \,|\, (Y \in X)}{1 + \sum_{i=0}^{N} i \,|\, (F \notin X)}, \qquad (15)$$

где j – количество обращений к некоторому методу, структуре, интерфейсу или перечислению Y, принадлежащим данному классу X.

В знаменателе мы добавляем 1 для того, чтобы предотвратить деление на 0 в случае, если окажется, что класс не использует ни одного метода, интерфейса, структуры, перечисления из других классов. В формулах (14) и (15) F и Y – это указание на все встречающиеся в классе методы, структуры, интерфейсы, перечисления, используемые при подсчете их количества.

В случае вычисления величины ABC_X^Y следует учитывать, что в случае, если данное значение будет равно числителю, то это означает, что класс не использует внешних методов, интерфейсов, структур и перечислений - и это является наилучшим показателем. В случае, если значение $ABC_X^Y < 1$, то это означает, что класс использует внешних объектов существенно больше, чем своих собственных, т.е. это - класс-посредник, и чем меньше это значение, тем в большей степени выражено посредничество данного класса. Если же значение $ABC_X^Y > 1$, то этот класс уже не посредник, и чем больше ABC_X^Y стремится к значению числителя, тем менее этот класс использует методы, интерфейсы, структуры и перечисления из других классов, а значит, этот класс сам выражает собственную абстракцию и описывает какой-то объект.

Для каждого конкретного исходного кода данная метрика будет всегда иметь точное значение верхней границы, равное количеству классов, интерфейсов, структур или перечислений в зависимости от того для какого из этих типов данная метрика вычисляется.

2. Обобщенная оценка качества исходного кода

Одной из важных задач в анализе исходного кода является получение обобщенной оценки качества исходного кода. Благодаря наличию нормированных метрик исходного кода можно построить обобщенную оценку исходного кода.

Пусть есть n метрик m_i. Для каждой метрики m_i вычисляется ее значение x_i. В n-мерном пространстве строится вектор, каждой координатой которого является значение метрики $(x_1,x_2,\ldots,x_n)$. Длина такого вектора в n-мерном Эвклидовом пространстве и является комплексной оценкой состояния исходного кода в момент его анализа:

$$g(x) = \sqrt{\sum_{i=1}^{n} x_i^2}, \qquad (21)$$

где n – количество метрик; x_i – значение i-й метрики.

Максимальная длина такого вектора равна длине вектора, каждая координата которого равна 1. Таким образом

$$g_{max}(x) = \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{n * 1} = \sqrt{n}. \qquad (22)$$

Полученную комплексную оценку необходимо нормировать, чтобы ее значение попадало в интервал [0–1]. Для этого применяется формула

$$\bar{g}(x) = \frac{g(x)}{|\bar{x}|_{max}} = \frac{g(x)}{\sqrt{n}} = \sqrt{\sum_{i=1}^{n} \frac{x_i^2}{n}}. \qquad (23)$$

Для того чтобы указать важность той или иной метрики, можно использовать вес метрики a_i. Каждый вес задается числом в пределах [0–1]. Можно использовать предопределенные коэффициенты, представленные в текстовом виде, например, 0 – метрика не задана, 0,25 – учитывать, но не принимать во внимание, 0,5 – метрика имеет низкую важность, 0,75 – обратить внимание, 1 – метрика имеет большую важность. На веса необходимо наложить ограничение $\sum_{i=1}^{n} a_i = 1$. Тогда веса должны быть нормированы.

Нормирование весов выполняется следующим образом:

$$a'_k = a_k \left(\sum_{i=1}^{n} a \right)^{-1}. \quad (24)$$

В результате, E(x) – оценка качества исходного кода с учетом весовых коэффициентов важности метрик, может быть вычислена по формуле:

$$E(x) = \sqrt{\sum_{i=1}^{n} \frac{a'_i x_i^2}{n}}. \quad (25)$$

Таким образом, мы получили среднеквадратичную взвешенную эвклидову характеристику исходного кода в некоторый момент времени t. Наличие такой обобщенной оценки позволяет строить алгоритмы сравнения двух и более версий исходного кода и показывать, каким образом меняется качество исходного кода с точки зрения метрик. Более того, появляется возможность оценить правильность направления разработки исходного кода с точки зрения его приближения или удаления от идеального состояния определяемого по формуле (22).

Выводы

В результате исследований, изложенных в статье, выполнены следующие задачи:

1. Сформирована система нормированных метрик исходного кода.

2. Разработан алгоритм вычисления обобщенной оценки качества исходного кода.

Дальнейшие исследования направлены на разработку алгоритмов анализа UML диаграмм, что позволит исправлять ряд ошибок на уровне архитектуры еще до того, как будет реализован исходный код. Также будут продолжены исследования в направлении изучения связи метрик между собой, их связи с такими свойствами исходного кода как надежность, повторное использование, расширяемость и т.д.

Список литературы

1. DeMarco Tom. Controlling Software Projects: Management, Measurement, and Estimates / T. DeMarco. – Prentice Hall, ISBN 0131717111, 1986.

2. Chidamber Shyam R. A Metrics Suite for Object Oriented Design / R. Chidamber Shyam. – New Jersey, Prentice-Hall, Inc, 1994.

3. DeMarco Tom. Software Engineering: An Idea Whose Time Has Come and Gone? / T. DeMarco // IEEE Software (ISSN 0740-7459). – July/August 2009. – P. 95-96.

4. Jagdish Bansiya., A Hierarchical Model for Object-Oriented Design Quality Assessment / Jagdish Bansiya, Carl G.Davis. // IEEE Transactions of software engineering. – Vol. 28, NO.1. – January 2002.

5. Lipanov A. Expert System for Software Source Code Quality Analysis / A. Lipanov, N. Vegerina // 2010 6th Central and Eastern European Software Engineering Conference (CEE-SECR). IEEE Catalog Number: CFP1CER-ART. ISBN: 978-1-4577-0606-6. – P. 86-92.

6. Pressman, Roger S. Ph.D (2001). Software Engineering - A Practitioner's Approach - Fifth Edition / Pressman, Roger S. – ISBN 0-07-365578-3, 2001.

Поступила в редколлегию 4.07.2011

Рецензент: д-р техн. наук, проф. Е.П. Путятин, Харьковский национальный университет радиоэлектроники, Харьков.

МЕТОД ПОБУДОВИ УЗАГАЛЬНЕНОЇ ОЦІНКИ ЯКОСТІ ПОЧАТКОВОГО КОДА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

О.В. Ліпанов

У статті розглядається новий підхід до отримання узагальненої оцінки якості початкового кода програмного забезпечення (ПО) з використанням метрик. У роботі приводиться аналіз і модифікація метрик початкового кода ПО для набуття їх нормованого значення в інтервалі [0, 1]. Переклад метрик в метричний простір дозволяє розглядати їх як деяку групу, в якій є певні зв'язки і чинники взаємного впливу метрик один на одного. Аналіз такого простору метрик дозволяє говорити про заходи близькості, аналіз збіжності або розбіжності метричних груп. Це дозволяє отримати комплексну оцінку якості початкового кода на підставі групи метрик, а не однієї конкретної метрики. Інтегральна оцінка початкового кода дозволяє здійснювати швидкий контроль напряму розвитку початкового кода.

Ключові слова: *метрики початкового кода, єдність, зчеплення, нестабільність, відносна єдність, асоціація між класами, абстрактність, нормування метрик, експертна система аналізу початкового кода.*

METHOD FOR BUILDING INTEGRATED ESTIMATION FOR SOFTWARE SOURCE CODE QUALITY

A.V. Lipanov

In this paper a new approach for obtaining of integral estimate of software source code with using metrics was described. In this paper described analysis and modifications of source code metrics for obtaining their normalized value within the range [0, 1]. Conversion of metrics into metric space allows treating them as some group with certain connections and factors of metrics' interference. Analysis of spaces allows talking about measures of proximity, analysis of convergence or divergence of metric groups. This allows obtaining integrated assessment of source code on basis of metrics' group, and not on one certain metric. Source code integrated estimate allows conducting quick control of source code development direction.

Keywords: *source code metrics, cohention, coupling, instability, relational cohention, association between classes, abstractness, metrics normalization, expert system for source code analysis.*