

Expert system for software source code quality analysis

Abstract

In this scientific report the expert system of source code quality analysis using metrics calculation is being described. Modern software products provide functionalities for counting source code metrics, but their values are non-normalized and don't have the limits of measurement. There are no software products that can interpret the numerical values of metrics to verbal recommendations up to date. The system is designed as a add-in and extension to the existing program Reflector. The system provides the user the ability to load module for the analysis, calculate metrics, get text recommendations and generate report. The system can be used in testing and software support. The system is developed using .NET Framework 3.5 and its further development will take place towards the development of independent software that can analyze the source code developed in C++, C#, Java and PHP.

Keywords: *object-oriented metrics, coupling, cohesion, instability, source code review.*

Экспертная система анализа качества исходного кода программного обеспечения

Реферат

В докладе рассматривается экспертная система оценки качества исходного кода с помощью подсчета метрик и выдачи текстовых рекомендаций. Современные средства предоставляют возможности для подсчета метрик по исходному коду, однако выдаваемые ими значения являются ненормированными и без указания пределов измерения. Средства, интерпретирующие числовые значения метрик в словесные рекомендации на сегодняшний день не существует. Данная система может быть применена при тестировании и поддержке программного продукта для рефакторинга, реинжиниринга и уменьшения количества уязвимостей исходного кода. Система разработана как дополнение и расширение к существующей программе Reflector. В системе предусматриваются возможности загрузки пользователем модуля для анализа, проводимого с помощью функциональности программы Reflector, подсчета метрик, выдачи на основе их значения текстовых рекомендаций с помощью экспертной системы и генерации отчета. Систему можно использовать для анализа исходного кода и получения текстовых рекомендаций по его модификации. Система разрабатывается с использованием технологии .NET Framework 3.5 и дальнейшее ее развитие будет происходить в направлении разработки самостоятельного программного продукта, который сможет анализировать исходный код разработанный на C++, C#, Java и PHP

Ключевые слова: *объектно-ориентированные метрики, связность, сцепление, нестабильность, анализ исходного кода.*

1. Введение. Современные средства проектирования и анализа исходного кода ПО

Проблема разработки надежного, масштабируемого, безопасного и работающего с высокой скоростью программного продукта является актуальной, несмотря на огромный прогресс, происходящий в сфере разработки программного обеспечения.

Новые методологии разработки, такие как экстремальное программирование или Scrum, позволяют сократить время разработки, а наличие новых платформ и абстрагирование от нижних уровней позволяют избегать многих ошибок. Тем не менее, контроль качества должен осуществляться на самых различных уровнях – начиная с методологического и заканчивая технологическим уровнем, когда процессы контроля качества протекают в автоматическом

режиме, например при автоматических сборках проекта. Любой контроль предполагает наличие метрик, которые позволяют оценить достижение того или иного уровня качества программного проекта. Применительно к контролю качества исходного кода можно использовать метрики кода. Они разделяются на категории и могут помочь оценивать различные аспекты программной системы: сложность и структурированность программного кода, связность компонентов, относительный объем программных компонентов и др.

Том ДеМарко, разработавший идею метрик исходного кода сказал: «Мы не можем управлять тем, что мы не можем измерить». Благодаря ему, оценка кода свелась к получению набора значений некоторых метрик, описывающих данный код [1]. На данный момент это единственный численный метод оценки исходного кода.

Существует большое количество метрик, однако многие из них либо являются сложными в вычислении, либо несут ненужную или противоречивую информацию о состоянии качества исходного кода. Метрики кода могут служить также для выявления архитектурных особенностей. Наибольший эффект применение таких метрик дает при анализе больших программных систем, когда ручной анализ и просмотр исходного кода может занимать значительное время.

Метрики программного кода являются важным инструментом и уже сегодня используются многими производителями программного обеспечения. Так, при сертификации на более высокие уровни по моделям ISO/IEC или CMM/CMMI использование метрик кода является обязательным, что позволяет в определенной степени достичь контролируемости процесса разработки [2].

Начальным этапом создания программного продукта является проектирование и разработка его архитектуры. Это наиболее важный момент всего процесса разработки приложения, так как изначально неправильно спланированная архитектура может привести к большому количеству ошибок, обнаруживаемых на стадии тестирования, а также ненадежной и небезопасной работе всего приложения. Для решения такой проблемы были созданы средства проектирования, такие как Rational Software Architect, Borland Together, ArgoUML, PowerDesigner и ряд других. Все эти продукты предоставляют очень широкий спектр функций, необходимых для проектирования ПО, в том числе использование языка UML, автоматическую генерацию исходного кода, учет ошибок и многие другие функции. Если в процессе создания приложения не использовать указанные системы, то это может привести к дополнительным временным затратам на доработки, изменение и тестирование [3].

Также следует отметить средства для анализа исходного кода, которых на сегодняшний день существует сравнительно небольшое количество. Они позволяют подсчитывать объектно-ориентированные метрики по исходному коду. Самыми известными программными продуктами, предоставляющими возможность подсчета наиболее оптимальных метрик, являются NDepend, OxyProject Metrics, Resource Standard Metrics, Visual Studio 2010, CodeMaid, CodeSnippets, GMetrics, SourceAnalyzer, Software Index, Resource Standard Metrics. В этих продуктах реализованы различные функции, которые выполняют подсчет различных метрик, например, количество строк (общее количество

строк, количество физических, логических, закомментированных, пустых, эффективных строк кода), метрики методов (количество входных и выходных параметров, цикломатическая сложность, функциональная и интерфейсная сложность), количество сборок, классов (в том числе абстрактных, исключений, перечислений, атрибутов и делегатов), интерфейсов, типов (в том числе значимых, с модификатором public и обобщений), методов (с модификатором public и обобщенных); количество свойств, методов и аргументов методов в интерфейсах и классах, а также цикломатическая сложность неабстрактных методов. Эти системы позволяют проводить анализ программ разработанных на C, C++, C#, JAVA, VB. Также есть системы, которые позволяют вычислять метрики по UML-диаграмме классов и они не зависят от языка программирования. Примером такой системы является система SDMetrics, которая предоставляет возможность посчитать более 100 различных метрик в таких категориях, как размер, наследование, связность, сцепление (реляционная сплоченность), сложность. SDMetrics реализует в несколько раз больше метрик, чем другие программы и, таким образом, позволяет наиболее полно оценить архитектуру приложения и выдать необходимые рекомендации.

Системы проектирования ПО эффективны лишь на первом шаге написания приложения. Если учесть, насколько может измениться структура приложения в процессе разработки (не в последнюю очередь благодаря изменению требований) либо в процессе поддержки, когда отладкой ошибок занимается команда разработчиков, не имевших отношение к написанию данного продукта, которым, однако, необходимо уменьшить количество уязвимостей в программном продукте, то средства проектирования теряют свою актуальность, и возникает необходимость в оценке состояния качества готового исходного кода. Одним из решений данной проблемы является применение метрического анализа, основанного на вычислении объектно-ориентированных метрик. Под вышесказанным подразумевается анализ внутренней объектной структуры исходного кода, отражающей сложность каждой отдельно взятой сущности, такой, как метод или класс, и внешней структуры, отражающей сложность взаимодействия сущностей между собой (например, связность или наследование) [2].

Приложения, которые вычисляют метрики, дают их ненормированные числовые значения без указания пределов измерения. Проще говоря,

если пользователю неизвестен смысл метрики, то ее значение не представляет для него никакого интереса. Более того числовые значения метрик сами по себе не несут никакой полезной информации. Это приводит к проблеме интерпретации значений метрик и перевода их в осмысленные утверждения о состоянии качества кода и рекомендации по его улучшению.

Среди существующих программных продуктов, нацеленных на проведение в той или иной степени оценки качества исходного кода, не существует приложений, предоставляющих возможности интерпретации метрик, с целью предоставления словесных рекомендаций по улучшению исходного кода. В связи с этим, проведение исследований в направлении разработки инструмента для оценки качества исходного кода с возможностью предоставления словесных рекомендаций является актуальной задачей.

В качестве основных задач разработки продукта определены были следующие задачи:

- нормировать выбранные метрики таким образом, чтобы их значения попадали в промежуток $[0, 1]$ для четкого определения уровня качества исходного кода;

- для каждой метрики разбить промежуток $[0, 1]$ на части и поставить в соответствие каждой части некоторую рекомендацию, отражающую состояние проверяемого кода и содержащую рекомендацию по его улучшению в случае необходимости;

- реализовать алгоритм подсчета каждой отдельной метрики;

- произвести анализ большого количества примеров исходного кода экспертами с целью получения оценки каждого примера и рекомендаций по его улучшению, а также сгруппировать полученные оценки по подобности примеров;

- реализовать экспертную систему, выдающую рекомендации на основе каждой метрики в зависимости от ее значения;

- разработать приложение, производящее разбор исходного кода, подсчитывающее метрики и выдающее текстовый эквивалент их значений, а также имеющее интуитивный дружественный интерфейс.

2. Метрики, реализованные в автоматизированной системе анализа исходного кода ПО

Существует множество различных классификаций метрик программного обеспечения, трактующих метрики с различных позиций и ранжирующих одни и те же характеристики по различным критериям. Одной из таких классификаций может служить разделение

метрик на группы по субъектам оценки:

- размер – сравнительная оценка размеров ПО;
- сложность – оценка архитектуры и алгоритмов программной системы (отрицательные показатели этой группы метрик говорят о проблемах, с которыми можно столкнуться при развитии, поддержке и отладке программного кода);

- поддерживаемость – оценка потенциала программной системы для последующей модификации.

Для оценки и контроля качества кода могут непосредственно использоваться метрики сложности: цикломатическая сложность, связность кода, глубина наследования и др.

Разработанная система предоставляет возможность оценки состояния качества исходного кода. Данная функциональность была реализована с использованием метрического метода, для которого были отобраны три метрики – **сцепление, связность и нестабильность**, которые наиболее объективно оценивают исходный код. Подсчет вышеуказанных метрик возвращает числовые значения, которые несут сравнительно небольшое количество информации для пользователя. Для решения данной проблемы была использована экспертная система. Таким образом, пользователь программы получает текстовую информацию о состоянии кода, а также рекомендации по его улучшению, если таковые необходимы.

Данная программа была разработана как дополнение и расширение к программе Reflector от компании Red Gate, являющейся бесплатной утилитой для Microsoft .NET, комбинирующей браузер классов, синтаксический анализатор и декомпилятор.

Текущая версия системы поддерживает только .NET модули в силу того, что программа Reflector позволяет получить исходный код только .NET модулей. После выбора файла производится анализ модуля путем подсчета метрик. Алгоритм подсчета каждой метрики реализован в отдельном классе, являющемся наследником класса Metric. Отображение результатов каждой метрики происходит в отдельном контроле, который является наследником MetricControl. Такая архитектура позволяет быстро и легко добавлять реализацию подсчета новых метрик в программу, и, кроме того, добавление метрик никаким образом не затрагивает существующий код и не приведет к появлению ошибок.

Также данный элемент управления осуществляет вывод текстовой информации, полученной в результате анализа. Данная функциональность реализована с помощью экспертной системы.

Каждая метрика имеет некоторый диапазон значений от минимального до максимального. Этот диапазон разбивается на несколько промежутков, для каждого из которых, в зависимости от метрики, была составлена рекомендация на основе оценок экспертов. Эти рекомендации и промежутки, к которым они относятся, хранятся в отдельном файле формата `nv`, основанном на формате `xml`. В процессе анализа программа подсчитывает значение метрики, загружает все промежутки, на которые был разбит диапазон значений метрики, из файла и в зависимости от того, в какой промежуток попадает значение метрики, выбирает необходимую рекомендацию, которая выводится пользователю. Такое решение позволяет убрать зависимость между подсчетом метрики и текстом рекомендации, так как рекомендации могут быть добавлены отдельно, равно как и промежутки диапазона значений. На данный момент файл с рекомендациями является не редактируемым, однако такая функциональность может быть добавлена в программу при необходимости.

2.1 Сцепление

В компьютерных науках сцепление (*coupling*) или зависимость – это мера того, насколько программный модуль зависит от каждого из остальных модулей.

Сцепление может быть низким (свободным, слабым) или высоким (плотным, сильным). Типы сцепления в порядке от самого высокого до самого низкого следующие (рис. 1) [4]:

– сцепление по содержимому (высокое) – один модуль изменяет или зависит от внутренней работы другого модуля (имеет доступ к локальным данным другого модуля). Изменения в вычислении данных во втором модуле повлекут

за собой изменения в зависимом модуле;

– совместное сцепление – два модуля разделяют одни и те же глобальные данные. Изменения в глобальных данных приведут к изменениям во всех модулях, использующих эти данные;

– внешнее сцепление – возникает, когда два модуля разделяют внешний установленный формат данных, протокол сообщений или интерфейс устройства;

– сцепление управления – один модуль контролирует выполняющий поток другого, передавая ему информацию о том, что необходимо выполнить;

– сцепление по образцу – один модуль посылает другому составную структуру данных в то время, как тому необходима только часть этой структуры. Это может привести к изменению чтения модулем структуры, так как поле, в котором модуль не нуждается, было изменено;

– сцепление данных – модули разделяют данные, например, по параметрам;

– сцепление по сообщениям – модули не зависят друг от друга. Вместо этого они используют интерфейс для обмена сообщениями без параметров (или событиями);

– нет сцепления – модули не зависят друг от друга.

Сцепление между классами А и В возрастает, если [4]:

– А содержит атрибут типа В;

– А вызывает метод (событие и др.) объекта В;

– А содержит метод, который ссылается на В (через возвращаемый тип или параметр).

Низкое сцепление представляет собой связь, при которой один модуль взаимодействует с другим с помощью простого и устойчивого интерфейса и не имеет необходимости знания о внутренней реализации другого модуля.

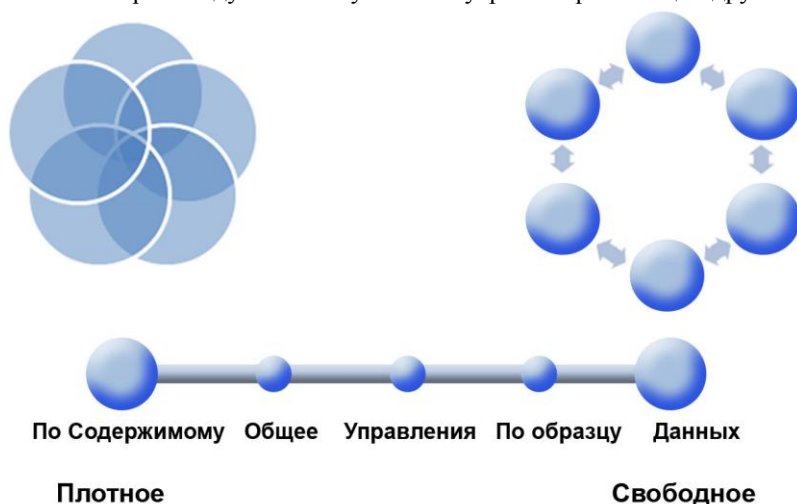


Рис. 1 – Концептуальная модель сцепления

Формула для вычисления сцепления имеет вид:

$$Coupling = r_t \left(\sum_{i=0}^n r_i \right)^{-1}$$

где r_t – количество ссылок на класс, где ссылка – это поле, локальная переменная, возвращаемый тип или параметр метода;

r_i – количество ссылок на класс, участвующий в подсчете метрики,

n – количество классов.

Значение сцепления тем больше, чем больше модуль сцеплен и находится в пределах приблизительно от 0.67 (низкое сцепление) до 1 (высокое сцепление).

2.2 Связность

В компьютерной науке связность (cohesion) – это мера того, насколько сильно связаны между собой методы модуля приложения. Связность является обычным типом измерения и чаще всего выражается как "высокая связность" или "низкая связность". Модули с высокой связностью являются более предпочтительными, так как высокая связность связана с некоторыми желательными характеристиками программного обеспечения, такими, как устойчивость, надежность, повторное использование и понятность, в то время как низкая связность ассоциируется с нежелательными особенностями, такими, как сложность поддержки, тестирования, повторного использования и даже сложность понимания.

Связность часто является противоположностью сцеплению: высокая связность соответствует низкому сцеплению и наоборот. Метрики контроля качества исходного кода сцепления и связности были разработаны Ларри Костантином и основаны на характеристиках "хорошей" программной практики, которая уменьшает расходы на поддержку и модификации. Таким образом, связность уменьшается, если [4]:

- методы класса мало связаны между собой;
- внутри методов выполняется много различных действий, использующих не связанные наборы данных.

Недостатки низкой (слабой) связности [4]:

- возрастание сложности понимания модулей;
- возрастание сложности поддержки системы, так как логические изменения в одном модуле влияют на многие другие модули и, таким образом, изменения в одном модуле влекут за собой изменения в связанных модулях;
- возрастание сложности повторного использования модуля, так как большинству приложений не нужны произвольные наборы операций, предоставляемые модулем.

Типы связности в порядке возрастания от низкой к высокой следующие:

– случайная связность (низкая) – части модуля связаны произвольно либо не имеют важной связи (модули часто используемых функций);

– логическая связность – части модуля сгруппированы, так как они логически выполняют одно и то же, даже если они разные (группы методов ввода/вывода);

– временная связность – части модуля группируются во время выполнения: части модуля обрабатываются в определенное время выполнения программы (метод, вызываемый после обработки исключения, который закрывает открытый файл, создает лог ошибок и оповещает пользователя);

– процедурная связность – части модуля сгруппированы, потому что они всегда следуют определенной последовательности выполнения (метод, проверяющий наличие разрешений и затем открывающий файл);

– коммуникационная связность – части модуля сгруппированы, так как они работают с одними и теми же данными (модуль, который обрабатывает одну и ту же информационную запись);

– последовательная связность – части модуля сгруппированы, так как результат выполнения одной части является входным параметром для другой части (метод, который читает данные из файла, и метод, обрабатывающий эти данные);

– функциональная связность (высокая) – части модуля сгруппированы, так как они сотрудничают при выполнении одной хорошо-определенной задачи модуля (например, разбор XML в случае Expat(XML)).

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

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

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

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 экземпляльному полю,

$$\sum_{i=0}^n MF - \text{сумма MF по всем экземплярам}$$

полям класса.

Основная идея метрик заключается в том, что класс абсолютно связан, если все его методы используют все его поля, что означает, что $\text{sum}(MF) = M * F$ и приводит к тому, что $\text{LCOM} = 0$ и $\text{LCOM HS} = 0$;

Классы, в которых $\text{LCOM} > 0$, количество полей больше 8 и количество методов больше 8, являются проблематичными. В тоже время сложно избежать таких несвязных классов. Классы, где $\text{LCOM HS} > 1$, количество полей больше 10 и количество методов больше 10, должны избегаться. Вышеупомянутое ограничение является более строгим, чем первое [4].

2.3. Нестабильность

Нестабильность (Instability) – это мера того, насколько зависят классы в пределах модуля. Однако, в отличие от сцепления, где подсчитывалась зависимость между двумя классами, нестабильность подсчитывает, насколько класс зависит от всех классов модуля в целом и насколько все классы модуля в целом зависят от данного класса. Для подсчета данной метрики производится подсчет связанных метрик: центро-стремительного и центробежного сцепления [5].

Центростремительное сцепление подсчитывает количество классов, которые зависят от данного класса. Центробежное сцепление, наоборот, считает количество классов, от которых зависит данный.

Формула для подсчета не стабильности выглядит следующим образом [5]:

$$I = \frac{C_e}{C_a + C_e},$$

где C_e – центростремительное сцепление,

C_a – центробежное сцепление.

$I = 0$ указывает максимально стабильный класс. $I = 1$ указывает максимально нестабильный класс. Большое значение метрики означает, что небольшие изменения в классах, от которых зависит данный, приведут к большим изменениям в данном, а также изменения в данном классе приведут к изменения в классах, которые зависят от данного.

3. Реализация системы анализа исходного кода ПО

Для работы программы, необходимо запустить программу Reflector с установленным модулем NVMetrics. Элемент управления NVMetrics состоит из четырех основных частей: панели управления, элемента управления с информацией о загруженном модуле, вкладок с элементами управления, подсчитывающими метрики и отображающими результаты, и элемента управления, отображающего рекомендации по выбранному значению метрики для класса (рис. 2).

Панель управления состоит из трех кнопок: Open, Analyze и Get report. Open позволяет выбрать dll-модуль для анализа, после чего в верхней части основного элемента управления появляется информация о загруженном модуле: название и путь к файлу. Нажатие на кнопку Analyze приводит к анализу выбранной сборки.

Анализ состоит из двух частей. Сначала программа получает полную внутреннюю структуру модуля с помощью функциональности Reflector и заполняет свою собственную структуру кода.

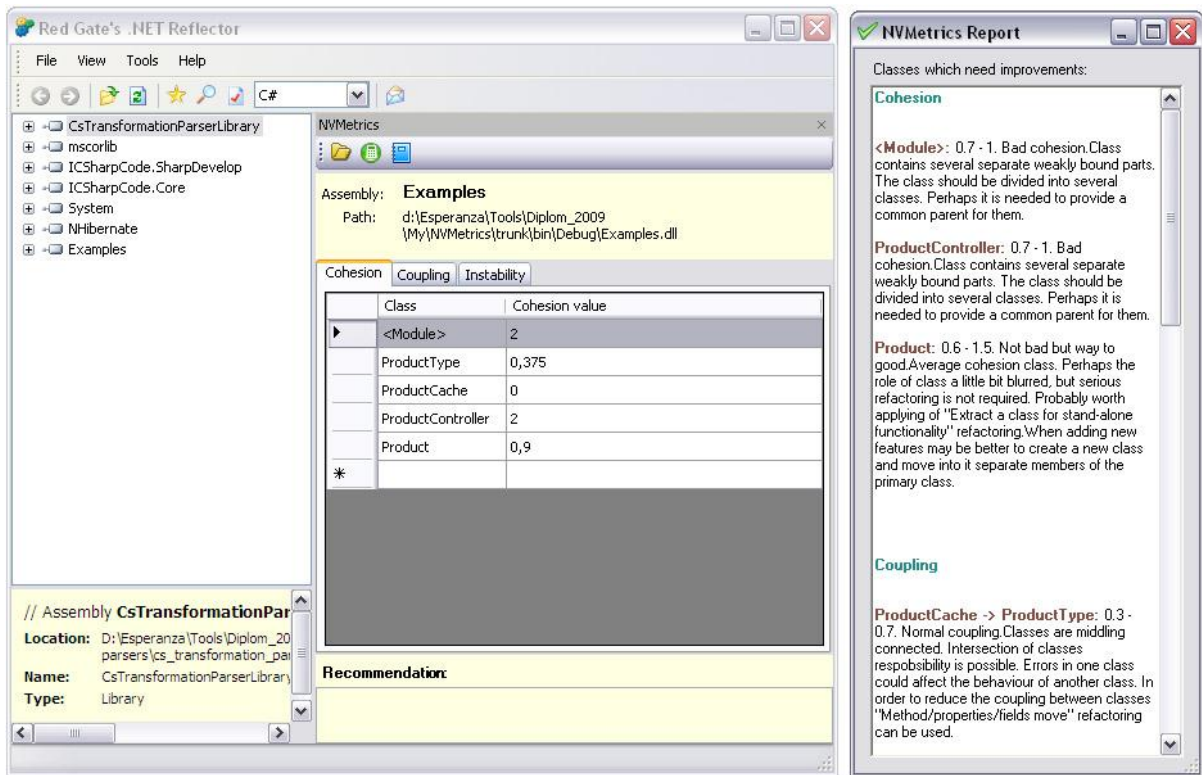


Рис. 2 – Отображение результатов подсчета метрик и рекомендации по модификации исходного кода

Затем на основе имеющей структуры производится подсчет трех метрик, описанных в п.2.1-2.3. Данные метрики вычисляются независимо друг от друга. Алгоритм подсчета и отображение результатов максимально отделено для уменьшения количества возможных ошибок.

Результаты анализа отображаются на вкладках. Каждая вкладка представляет собой элемент управления, в котором находится список всех классов модуля, а также вычисленные значения метрик в виде списка (Cohesion и Instabilty) или таблицы (Coupling). При нажатии на ячейку со значением метрики в нижней части основного элемента управления появляется текстовая информация о состоянии и качестве кода, а также рекомендации по его улучшению, если таковые необходимы.

4. Заключение

Тестирование системы показало ее пригодность к использованию для анализа исходного кода и адекватность выдаваемых рекомендаций по модификации исходного кода, а следовательно подход реализованный в системе является правильным. Дальнейшее развитие системы будет происходить в направлении разработки самостоятельного программного продукта, который сможет анализировать исходный код разработанный на C++, C#, Java и

PHP. Также будет расширяться список метрик, используемых при анализе исходного кода и формулироваться рекомендации соответствующие выбранным метрикам. В качестве дополнительной возможности будет реализована возможность генерации UML диаграмм для анализируемого исходного кода, что позволит постоянно поддерживать документацию по проекту в актуальном состоянии.

ПЕРЕЧЕНЬ ССЫЛОК

- [1] DeMarco Tom. Controlling Software Projects: Management, Measurement, and Estimates. Prentice Hall, ISBN 0131717111, 1986.
- [2] Совершенный код. Мастер-класс / Пер. с англ. – М.: Издательско –торговый дом «Русская редакция»; СПб.: Питер, 2005. – 896 с.: ил.
- [3] Буч Г., Якобсон А., Рамбо Дж. UML. Классика CS. 2-е изд./ Пер. с англ.; Под общей редакцией проф. С. Орлова - СПб.: Питер, 2006. – 736 с.: ил.
- [4] Chidamber Shyam R. A Metrics Suite for Object Oriented Design. / Chidamber Shyam R., Kemerer Chris F. – New Jersey, Prentice-Hall, Inc, 1994.
- [5] Martin, Robert C. Designing object-oriented C++ applications using the Booch method / Martin, Robert C.–New Jersey, Prentice-Hall, Inc, 1995.–530p.