# Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет

Информатики и радиоэлектроники”

РЕФЕРАТ

Метрология, стандартизация и сертификация

в информационных технологиях

Практическая реализация метрик кода в IDE

Выполнил:

Студент гр. 020601

Шумигай В.В.

**Минск 2021**

СОДЕРЖАНИЕ

Введение…………………………………………………………………………...3

[Создание данных метрик кода 4](#_Toc86348119)

[Правила метрик кода для анализаторов качества кода .NET 4](#_Toc86348120)

[Конфигурация 4](#_Toc86348121)

[Использование окна "Результаты метрик кода" 5](#_Toc86348122)

[Отображение результатов метрик кода 5](#_Toc86348123)

[Фильтрация результатов метрик кода 5](#_Toc86348124)

[Применение фильтров к результатам метрик кода 6](#_Toc86348125)

[Добавление, удаление, изменение порядка столбцов данных 6](#_Toc86348126)

[Добавление или удаление столбца 6](#_Toc86348127)

[Переупорядочение столбцов 6](#_Toc86348128)

[Копирование данных в буфер обмена или Excel 7](#_Toc86348129)

[Создание рабочего элемента на основе результатов метрик кода 7](#_Toc86348130)

[Создание рабочего элемента на основе результата 7](#_Toc86348131)

[Создание ошибки на основе результата 7](#_Toc86348132)

[Программные измерения 8](#_Toc86348133)

[Анонимные методы 9](#_Toc86348134)

[Созданный код 9](#_Toc86348135)

[Примеры метрик 10](#_Toc86348136)

[Цикломатическая сложность 10](#_Toc86348137)

[Метрика C.R.A.P. Гол 10](#_Toc86348138)

[Заключение 10](#_Toc86348139)

С развитием программных приложений увеличивается их сложность, а значит усложняется и сопровождение кода. Метрики кода представляют собой набор оценок программного обеспечения, благодаря которым разработчики получают более глубокое представление о разрабатываемом коде.

Метрики кода разделяются на категории и способны оценивать различные аспекты программной системы, а именно: сложность и структурированность программного кода, связность программных компонентов, относительный объем компонентов и др. Используя преимущества метрик кода, разработчики понимают, какие типы и методы должны быть переработаны или тщательно протестированы. Группы разработчиков могут выявлять потенциальные риски, понимать состояние проекта и отслеживать ход выполнения при разработке программного обеспечения.

Разработчики могут использовать разные метрики, основываясь на личном опыте или поставленной задаче. Однако при работе в команде все сложнее и любые изменения будут полезны только если:

* Команда разработала их
* Команда согласилась с ними
* Они используются для определения конкретной области

Иначе любая метрика пострадает от оптимизирующей команды.

Интегрированная среда разработки или *IDE* (англ. *Integrated development environment*) – комплекс программных средств, который используется разработчиками для создания программного обеспечения (ПО). *IDE* в первую очередь включают в себя: редактор исходного кода, компилятор и/или интерпретатор, инструменты автоматизации сборки и отладчик. Вышеперечисленные характеристики делают *IDE* профессиональными средствами для работы с ПО и их кодом соответственно.

Существует множество *IDE* со своими преимуществами и недостатками, но в качестве примера рассмотрим одну из наиболее известных и популярных, а именно *Visual Studio*. Разработчики могут использовать *Visual Studio* для создания метрик кода, которые измеряют сложность и удобство обслуживания кода.

## **Создание данных метрик кода**

Данные метрик кода можно создавать тремя способами:

* Включив анализаторы качества кода *.NET* и включив в него метрики кода.
* выбрав команду меню анализ > метрики кода в *Visual Studio*.
* из командной строки.

### **Правила метрик кода для анализаторов качества кода .NET**

Анализаторы качества кода .*NET* включают несколько правил анализатора кода метрик:

* CA1501
* CA1502
* CA1505
* CA1506

Эти правила по умолчанию отключены, но их можно включить в Обозревателе решений или в файле *EditorConfig*.

### **Конфигурация**

Вы можете настроить значения, при которых будут срабатывать правила метрики кода.

1. Создание текстового файла. Например, можно присвоить имя CodeMetricsConfig.txt.
2. Добавьте необходимые пороговые значения в текстовый файл в следующем формате:

*CA1502: 10*

В этом примере правило CA1502 срабатывает, когда сложность организации циклов метода больше 10.

1. В окне свойства Visual Studio или в файле проекта пометьте действие сборки файла конфигурации как аддитионалфилес. Пример:

*<ItemGroup>*

*<AdditionalFiles Include="CodeMetricsConfig.txt" />*

*</ItemGroup>*

## **Использование окна "Результаты метрик кода"**

В окне **Результаты метрик** **кода** отображаются данные, которые формируются анализом метрик кода.

### **Отображение результатов метрик кода**

Окно результатов метрик кода отображается автоматически при формировании результатов метрик кода. Окно также можно отобразить в любое время и любым из следующих способов:

* в меню анализ выберите ***Windows*** > **результаты метрик кода.**
* в меню вид выберите другие ***Windows* > результаты метрик кода.**

Откроется окно **Результаты метрик кода**, даже если в нем нет результатов.

## **Фильтрация результатов метрик кода**

Результаты, которые отображаются в окне **результатов метрик кода**, можно фильтровать при помощи панели инструментов вверху окна. Например, может понадобится просмотреть результаты с индексом ниже 65.

Список **Фильтр** содержит имена столбцов результатов. При определении фильтра он добавляется в конец списка вместе с отступом. Список способен содержать последние 10 фильтров, которые были определены.

### **Применение фильтров к результатам метрик кода**

1. В списке **Фильтр** выберите имя столбца.
2. В поле ***min* (минимум)** введите минимальное отображаемое значение.
3. В поле ***Max*** введите максимальное отображаемое значение.
4. Нажмите кнопку **Применить фильтр**.
5. Чтобы просмотреть сведения о результатах, разверните дерево иерархии.

## **Добавление, удаление, изменение порядка столбцов данных**

Столбцы «результаты» можно добавлять, удалять и изменять их порядок из окна «**Результаты метрик кода**».

### **Добавление или удаление столбца**

1. Выберите пункт **Добавить или удалить столбцы** или щелкните правой клавишей мыши заголовок столбца и выберите команду **Добавить или удалить столбцы**.
2. В диалоговом окне **Добавление или удаление столбцов** установите или снимите флажок напротив столбца, который требуется добавить или удалить, а затем нажмите кнопку ***ОК***.

### **Переупорядочение столбцов**

1. Нажмите кнопку **Добавить или удалить столбцы**.
2. В диалоговом окне **Добавление или удаление столбцов** выберите столбец, который необходимо переместить, а затем щелкните стрелку вверх или вниз.
3. Если столбец расположен в нужном месте, нажмите кнопку ***ОК***.

## **Копирование данных в буфер обмена или Excel**

Вы можете выбрать и скопировать строку данных метрик кода в буфер обмена как текстовую строку, содержащую одну строку для имени и значения каждого столбца данных. Можно нажать кнопку **открыть выделенный фрагмент в *Microsoft Excel***, чтобы экспортировать результаты метрик кода в электронную таблицу Excel.

## **Создание рабочего элемента на основе результатов метрик кода**

Можно создать *Azure Boards* рабочий элемент, основанный на результатах в окне **результатов метрик кода.** При создании рабочего элемента Visual Studio автоматически вводит заголовок в поле ***title*** и данные метрики кода на вкладке **журнал**.

### **Создание рабочего элемента на основе результата**

1. Щелкните результат правой кнопкой мыши.
2. Наведите указатель на пункт **создать рабочий элемент** и выберите тип рабочего элемента, который необходимо создать (**Ошибка, задача** и т. д.).
3. Заполните форму рабочего элемента.
4. В меню **файл** выберите команду **сохранить все**, чтобы сохранить рабочий элемент.

### **Создание ошибки на основе результата**

1. Выберите результат.
2. Нажмите **создать рабочий элемент**.
3. Заполните форму рабочего элемента.
4. В меню **файл** выберите команду **сохранить все**, чтобы сохранить рабочий элемент.

## **Программные измерения**

Результаты метрик кода, которые вычисляют в Visual Studio:

* **Индекс удобства обслуживания** — вычисляет значение индекса от 0 до 100, который показывает относительную простоту обслуживания кода. Высокое значение означает лучшую сопровождаемость. Для быстрого обнаружения проблем в коде можно использовать маркировку. Зеленая Оценка находится в диапазоне от 20 до 100 и означает, что код обладает хорошей сопровождаемостью. Желтая Оценка от 10 до 19 означает, что код является умеренно поддерживаемым. Красная Оценка — это оценка между 0 и 9 и указывает на низкую сопровождаемость.
* **Сложность организации циклов** — измеряет структурную сложность кода. Создается в результате вычисления количества различных путей кода в потоке программы. Программа, у которой сложный поток управления, требует больше тестов для достижения хорошего объема протестированного кода и менее сопровождаема.
* **Глубина наследования** — показывает количество различных классов, которые наследуются друг от друга, вплоть до базового класса. Глубина наследования аналогична взаимосвязанности классов в том, что изменение базового класса может повлиять на любые из его унаследованных классов. Чем больше это число, тем более глубокое наследование и тем выше вероятность изменения базового класса, что приводит к критическому изменению. Для глубокого наследования низкое значение хорошо, а высокое значение является недопустимым.
* **Взаимосвязь классов** — измеряет связь с уникальными классами через параметры, локальные переменные, возвращаемые типы, вызовы методов, универсальные экземпляры или шаблоны шаблонов, базовые классы, реализации интерфейса, поля, определенные во внешних типах, и декорирование атрибутов. Хорошая разработка программного обеспечения определяет, что типы и методы должны иметь высокую связность и низкую связь. Высокая связь указывает на проект, который трудно использовать и поддерживать из-за множества взаимозависимостей от других типов.
* **Строки исходного кода** — точное число строк исходного кода, имеющихся в исходном файле, включая пустые строки. Эта метрика доступна начиная с *Visual Studio* 2019 версии 16,4 и *Microsoft*. *CodeAnalysis*. метрик (2.9.5).
* **Строки исполняемого кода** — приблизительное количество строк или операций исполняемого кода. Это количество операций в исполняемом коде. Метрика доступна начиная с *Visual Studio* 2019 версии 16,4 и *Microsoft*. *CodeAnalysis*. метрик (2.9.5). Значение обычно является близким к предыдущей метрике (**строкам кода)**, которая является метрикой на основе инструкций языка *MSIL*, используемой в устаревшем режиме.

## **Анонимные методы**

**Анонимный метод** — это метод без имени. Анонимные методы чаще всего используются для передачи блока кода в качестве параметра делегата. Результаты метрик кода для такого метода, который был объявлен в элементе, например метод доступа, связаны с членом, который объявляет метод. Они не связаны с членом, который вызывает метод.

## **Созданный код**

Некоторые программные средства и компиляторы создают код, который добавляется в проект, и разработчик проекта не может видеть или не должен изменять его. В основном метрики кода игнорируют сформированный код при вычислении значений метрик. Это позволяет значениям метрик отражать, что может видеть и изменять разработчик. Код, созданный для *Windows Forms*, не пропускается, так как это код, который разработчик может просматривать и изменять.

## **Примеры метрик**

### **Цикломатическая сложность**

*Cyclomatic Complexity* (*CC*) Цикломатическая сложность метода — это число точек принятия решения в графе потока управления метода, увеличенное на единицу. Точки принятия решения возникают в операторах *if*/*for*/*while*, предложениях *case*/*catch* и аналогичных элементах исходного кода, где поток управления не просто линейный. Количество точек принятия решений (байт-кода), представленных одним оператором (исходного кода), может варьироваться в зависимости, например, от на сложность логических выражений. Чем выше значение цикломатической сложности метода, тем больше тестовых случаев требуется для проверки всех ветвей графа потока управления метода.

### **Метрика C.R.A.P. Гол**

Это алгоритм, который сравнивает взвешенную цикломатическую сложность с автоматическим тестированием. Алгоритм выглядит следующим образом: *CRAP(m) = comp(m)^2 \* (1 – cov(m)/100)^3 + comp(m)* где *comp (m)* - цикломатическая сложность метода *m*, а *cov (m)* - охват кода теста, предоставляемый автоматическими тестами. Метрическая система поощряет написание не сложного кода в сочетании с хорошим охватом тестами.

## **Заключение**

В настоящее время ни одной универсальной метрики не существует. Любые контролируемые метрические характеристики программы должны контролироваться либо в зависимости друг от друга, либо в зависимости от конкретной задачи, кроме того, можно применять гибридные меры, но они так же зависят от более простых метрик и также не могут быть универсальными. Строго говоря, любая метрика – это лишь показатель, который сильно зависит от языка и стиля программирования, поэтому ни одну меру нельзя возводить в абсолют и принимать какие-либо решения, основываясь только на ней.

Тем не менее метрики являются достаточно полезным инструментом в руках разработчиков и менеджеров проектов, позволяющим выявить моменты ухода разработки на более низкий качественный уровень и распознать наиболее сложные участки в системе. Определение числовых показателей может дать новые сведения о разрабатываемом продукте и помочь более грамотно планировать расходы на его дальнейшее развитие.