



Разработка программы для оценки показателей качества объектно-ориентированных приложений: обзор метрик, способов вычисления и инструментов для С

Контекст и постановка проблемы оценки качества ООП-приложений

Разработка программного обеспечения (ПО) практически всегда связана с решением сложных задач, для которых существует несколько альтернативных проектных и реализационных решений, отличающихся по скорости работы, потреблению ресурсов, сопровождаемости и другим характеристикам. В учебной презентации по метрикам эта ситуация формулируется как проблема выбора "наилучшего решения" среди альтернатив (пример с тремя алгоритмами, различающимися временем и памятью). 1

Ключевая причина, почему "выбор лучшего решения" нетривиален, связана с тем, что сложность ПО имеет много источников и не сводится к одному числу. В презентации отдельно упомянута идея Entity["people", "Фредерик Брукс", "software engineer"] о разделении сложности на **essential complexity** (неустранимая сложность, диктуемая самой предметной задачей) и **accidental complexity** (случайная/побочная сложность, возникающая из-за неудачных решений при проектировании и программировании). 2

В этом контексте **метрики ПО** выступают как формализованные численные измерения свойств кода/дизайна, которые позволяют: - сопоставлять варианты и результаты, - сравнивать с контрольными уровнями, - отслеживать динамику качества по мере развития системы. 1

При этом сама презентация подчёркивает важное ограничение: метрики не дают "объективной картины" автоматически; возможны эффекты оптимизации "под метрику" и формального использования без учёта контекста. 3

Именно поэтому раздел "Обзор по теме" для будущей ВКР должен не только перечислять метрики, но и содержать **критический анализ** того, *какие метрики выбрать, как именно их вычислять (варианты определений), какими средствами их можно получить, и почему выбранный подход рационален.*

Метрики из презентации и их интерпретация для ООП-качества

Ниже — метрики, присутствующие в предоставленной презентации, с группировкой по смыслу (акцент на ООП и архитектурные метрики).

Метрики наследования и “размерности” класса: - **DIT** (Depth of Inheritance Tree) — глубина в дереве наследования. ¹ - **NOC** (Number of Children) — число непосредственных потомков класса. ¹ - **NOM** (Number of Methods) — число локально определённых методов класса (без учёта видимости). ¹ - **WMC** (Weighted Methods Per Class) — сумма цикломатической сложности методов класса (в презентации веса = СС методов). ⁴

Метрики сложности управляемой логики: - **Cyclomatic Complexity** (CC) — число линейно независимых путей (в презентации дана формула через граф управления). ⁵ - **Cognitive Complexity** — когнитивная сложность (правила увеличения за “нарушения линейного чтения” и вложенность). ⁶

Метрики сцепления (coupling): - **CBO** (Coupling Between Object classes) — число связанных классов; в презентации связь включает вызовы, использование атрибутов, наследование и полиморфные вызовы. ¹
- **CF** (Coupling Factor) — нормализованное отношение числа клиентских отношений к числу возможных; вычисляется для всей программы, причём “клиентское отношение” не должно быть наследованием. ⁷ - **MPC** (Message Passing Coupling) — число “посыпаемых сообщений” (вызовов методов других классов), при этом вызовы собственных методов не учитываются. ³

Метрики связности (cohesion) класса: - Семейство **LCOM** (LCOM1...LCOM5) — несколько вариантов измерения отсутствия связности методов класса, от счёта пар методов до графовых вариантов и формульной нормализации. ⁸ - **TCC** (Tight Class Cohesion) и **LCC** (Loose Class Cohesion) — нормализованные доли связанных пар методов (TCC — только прямые связи, LCC — прямые и косвенные), с исключениями для конструкторов/деструкторов и некоторых методов. ⁹

Метрики архитектуры пакетов (в презентации “пакет = модуль/библиотека”): - примитивы: **NCP**, **OutC**, **InC**, **HC**, **SPC**, **SCC**. ¹⁰ - метрики по [entity["people", "Роберт Мартин", "software engineer"]]: **Ce** (efferent), **Ca** (afferent), **A** (abstractness), **I** (instability), **D** (distance). ¹¹

Критически важно отметить (и зафиксировать в постановке задачи ВКР), что уже на уровне “одного названия” метрика может иметь **несколько реализаций** (особенно LCOM), а разные инструменты нередко дают разные значения из-за различий в определениях и трактовках. Это отмечается и в исследованиях о несогласованности инструментов анализа (в т.ч. при оценке “технического долга” и связанных показателей) ¹² и в работах, где подчёркивается вариативность реализации LCOM/LOC/CC между инструментами. ¹³

Следствие для ВКР: программа должна **явно документировать** выбранные определения и (желательно) поддерживать режимы/варианты, совместимые с популярными инструментами.

Как вычисляются ООП-метрики для C# проектов

Ниже приведён практико-ориентированный обзор вычислений (чтобы затем можно было перейти к алгоритмам и коду). Акцент сделан на том, как метрики “из презентации” корректно формализовать для C#.

Метрики наследования и структуры класса

DIT (Depth of Inheritance Tree)

Определение в презентации: максимальная длина пути (в предках) от класса до корня дерева

наследования. ¹

Классическое определение СК-метрик: DIT — максимальная длина от узла до корня (в т.ч. при множественном наследовании — максимум). ¹⁴

Для C# (где множественного наследования классов нет) вычисление обычно сводится к: - построить цепочку `BaseType` до `System.Object` (или другого "корня" модели), - DIT = количество переходов по `BaseType`.

Практический нюанс для ВКР: решить, считать ли `System.Object` нулевым уровнем (DIT=0 для классов без явного базового класса) или единичным — в зависимости от выбранной конвенции. Презентация показывает пример $A \rightarrow B \rightarrow C$ без явного `Object`, что подразумевает "локальную" глубину. ¹

NOC (Number of Children)

Определение: число непосредственных потомков класса. ¹⁵

Для C# это: - для каждого класса собрать список классов, у которых `BaseType == данный класс`, - NOC = размер списка.

Важно документировать, включаются ли в "потомков" классы из других сборок/проектов (если анализируется только решение) и как учитываются `sealed` классы (обычно NOC=0).

NOM (Number of Methods)

Определение в презентации: число методов, определённых локально в классе, "без учёта видимости". ¹

Для C# главный методический вопрос — что считать "методом": - включать ли конструкторы, - включать ли аксессоры свойств (`get/set`) и событий (`add/remove`) как отдельные методы (многие анализаторы так делают, что заметно влияет на NOM и метрики связности).

Рациональная позиция для ВКР: задать **формальное правило счета** (например: считать `MethodDeclarationSyntax`, дополнительно `ConstructorDeclarationSyntax`, и отдельно описать обработку аксессоров), а затем обеспечить переключатели режима, если нужно сравнение с инструментами. ¹⁶

Метрики сложности управляемой логики

Cyclomatic Complexity (CC)

В презентации дана интерпретация CC как числа независимых линейных путей и формула через граф: $L - N + 2P$ (ребра, вершины, компоненты). ¹

Классическое инженерное обоснование и расчёт через CFG приведены в отчёте Watson & McCabe (NIST), где CC определяется через $e - n + 2$ для модулей (и связанный подход базисного тестирования). ¹⁷

Для C#-реализации в рамках ВКР наиболее воспроизводимый путь — строить control-flow graph (CFG) для метода и считать CC по графовой формуле или эквивалентным счётом "точек ветвления". Рекомендуемо учитывать, что в экосистеме .NET метрика CC активно используется и в правилах анализаторов (например, CA1502 в документации .NET объясняет смысл CC как числа независимых путей). ¹⁸

Cognitive Complexity

В презентации указаны базовые принципы: увеличивать на 1 за операторы, "прерывающие поток исполнения", и за вложенность; игнорировать "синтаксический сахар", повышающий читаемость. ¹

В спецификациях Sonar cognitive complexity формулируется как: - инкремент при каждом "разрыве линейного чтения" (циклы, условия, `catch`, `switch`, переходы к меткам, смешение логических операторов), - дополнительный инкремент за каждый уровень вложенности, - вызовы методов

сами по себе не увеличивают метрику ("method calls are free"). ¹⁹

Для C# это удобно реализуется синтаксическим обходом дерева (Syntax Walker) с учётом текущей глубины вложенности и специальных случаев (например, "цепочки условий", логические выражения, `goto / break label`, рекурсия, если выбран вариант расчёта с учётом рекурсивных вызовов). ²⁰

Метрики "веса" и сцепления на уровне класса

WMC (Weighted Methods Per Class)

В презентации WMC задана как сумма цикломатической сложности методов класса. ¹
В СК-метриках WMC определяется как сумма сложностей c_i методов класса; при $c_i = 1$ WMC становится просто числом методов. ¹⁴

Для ВКР важно зафиксировать, что "вес" равен **какой именно сложности**: - СС по CFG, - или иной "вес" (например, когнитивная сложность).

Презентация однозначно задаёт вариант "вес = СС". ¹

CBO (Coupling Between Object classes)

В презентации СВО трактуется как число связанных классов; связь включает вызов методов, использование атрибутов, наследование, полиморфный вызов. ¹

В СК-определении СВО — число других классов, с которыми класс сцеплён; сцепление возникает, когда методы одного используют методы/переменные другого. ²¹

Практическая вычислимость для C# требует решения двух методических вопросов: 1) что считать "использованием класса" (только в телах методов, или также в сигнтурах, базовых типах, атрибутах, generic-ограничениях, `typeof`, и т.п.);

2) считать ли наследование частью СВО (презентация — "да", СК-подход чаще измеряет coupling через `use/depends`, а наследование выделяется отдельными метриками DIT/NOC). ²²

Рациональная стратегия для ВКР: реализовать **две версии**: - CBO_{total} (как в презентации, включая наследование/полиморфизм), - $CBO_{no_inheritance}$ (для сопоставления с подходами, где inheritance исключают).

И в обоих случаях считать "число различных классов", а не "число обращений".

CF (Coupling Factor)

В презентации CF — нормализованное отношение числа клиентских отношений к числу возможных; подчёркнуто, что "другой класс не является предком" (т.е. inheritance исключается). ¹

Формула MOOD-подхода (в обзорных источниках по ОО-метрикам): CF выражается через отношение числа пар "client→supplier" к $TC^*(TC-1)$ (или эквивалентному выражению), где TC — число классов; связь определяется наличием хотя бы одной **не-наследственной** ссылки. ²³
На практике это означает: - построить ориентированный граф зависимостей классов, исключив наследование, - посчитать число различных пар (C_i, C_j) , $i \neq j$, для которых C_i зависит от C_j , - нормализовать на максимум $TC^*(TC-1)$. ²⁴

Метрики связности класса

LCOM1 и LCOM2

Презентация задаёт: - LCOM1 как число пар методов, не обращающихся к общему атрибуту, учитывая только собственные методы/атрибуты класса. ²⁵

- LCOM2 как $\max(P-Q, 0)$, где P — пары методов, обращающиеся к различным множествам переменных, Q — прочие пары. ³

Это согласуется с классическим определением СК-LCOM через множества instance-переменных методов и множества пар P/Q (разность $|P| - |Q|$, иначе 0). ²⁶

Алгоритмически для C#: 1) для каждого метода построить множество полей (и, по выбранному правилу, авто-свойств), к которым метод обращается, 2) для каждой пары методов проверить пересечение, 3) посчитать Р и Q и получить LCOM2; LCOM1 — частный счёт “непересекающихся пар”. ²⁷

LCOM3 и LCOM4

Презентация: - LCOM3 — число компонент связности в графе методов, где ребро есть, если методы имеют доступ к одному атрибуту. ²⁵

- LCOM4 — также число компонент, но с “учётом транзитивных связей”. ³

В исследовательских источниках по метрикам (например, в обзорах и инженерных спецификациях) LCOM4 обычно уточняется как граф, где методы связаны не только общими атрибутами, но и вызовами (call edges), и учитывается транзитивность по графу вызовов. ²⁸

Для C# практическое следствие: при вычислении LCOM4 необходимо добавить ребра “метод вызывает метод” (внутри класса) и, возможно, учитывать косвенное использование полей через цепочку вызовов.

LCOM5

В презентации LCOM5 задаётся через NOM, NOA и $\text{NOAcc}(m)$ (сколько атрибутов доступно методу). ²⁵

Эта формула соответствует нормализованному варианту семейства Henderson-Sellers, где используется среднее число “доступов методов к полям” (или эквивалентная сумма) и нормализация на (NOM-1). ²⁹

Реализация для C# требует аккуратно определить: - NOA: что считать “атрибутом” (поля, авто-свойства, backing fields), - доступность: считать ли чтение/запись, учитывать ли доступ через вызванные методы.

TCC и LCC

В презентации: - $TCC = NDC / NP$, $NP = N*(N-1)/2$, NDC — число **прямых** связей между парами методов через атрибуты;

- $LCC = (NDC + NIC) / NP$, где NIC — косвенные связи;

- исключаются конструкторы/деструкторы и ряд методов (интерфейсы/события). ³

В оригинальной работе Bieman & Kang TCC/LCC также определяются как доли непосредственно/непосредственно+косвенно связанных пар методов среди всех пар видимых методов (с обсуждением исключения конструкторов как “искусственного клея”). ³⁰

Для C# реализация обычно строится через: - моделирование связей методов по общим полям (и иногда по цепочкам вызовов), - подсчёт прямых и косвенных связей между парами, - нормализация на число пар. ³¹

Метрики архитектуры пакетов

Презентация вводит пакет как “модуль/библиотеку” и перечисляет примитивы NCP/OutC/InC/HC/SPC/SCC, а затем метрики Ce/Ca/A/I/D. ³

В инженерных руководствах по пакетным метрикам (например, deliverable консорциума SQUALE) эти примитивы определяются более строго: - **NCP** — число классов внутри пакета. ³²

- **OutC** — число классов пакета, имеющих исходящие зависимости на классы вне пакета. ³²

- **InC** — число классов пакета, имеющих входящие зависимости от классов вне пакета. ³²

- **HC** — число “скрытых” классов пакета без внешних связей. ³²

- **SPC** — сколько внешних классов использует пакет (зависимости пакета вовне, но на уровне классов). ³²

- **SCC** — сколько внешних классов зависит от пакета (зависимости на пакет вовне, но на уровне классов). ³²

Метрики Robert Martin (в презентации они приведены в "пакетной" версии): - **C_a** (afferent couplings) — число внешних классов, зависящих от классов пакета;
- **C_e** (efferent couplings) — число классов пакета, зависящих от внешних классов;
- **I = C_e/(C_a+C_e)**;
- **A = (#abstract)/(#total)**;
- **D** — расстояние до "main sequence" (в оригинале у Мартина есть вариант с делением на 2; в презентации — упрощённая форма $|A + I - 1|$). ³³

Для C# ключевой вопрос формализации: что считать "пакетом". Типичные варианты: - **assembly/project (.csproj)** как пакет (наиболее близко к "модуль/библиотека"), - **namespace** как пакет (ближе к логической декомпозиции). ³⁴

В ВКР разумно поддержать оба уровня: "пакет=проект" и "пакет=namespace", так как инструменты и команды разработки часто хотят видеть обе картины.

Анализ существующих инструментов, которые уже считают выбранные метрики

Одно из требований обзора — указать, какие средства разработки умеют считать какие метрики. Важно также учитывать, что "одна и та же" метрика в разных инструментах может иметь отличающиеся определения и область применения. ³⁵

Средства экосистемы .NET и Visual Studio

Code Metrics в Visual Studio / command-line Metrics.exe

Microsoft-экосистема предоставляет: - расчёт метрик из IDE, - расчёт из командной строки через `msbuild /t:Metrics`, - поставку `Microsoft.CodeAnalysis.Metrics` (NuGet) и возможность собрать `Metrics.exe` из исходников репозитория `dotnet/roslyn`. ¹⁶

Набор метрик, который выводит этот инструмент в отчёт (XML), включает по крайней мере: - Maintainability Index, - Cyclomatic Complexity, - Class Coupling, - Depth of Inheritance, - SourceLines / ExecutableLines. ³⁶

Сопоставление с метриками из презентации: - **Cyclomatic Complexity** — есть (соответствует CC).

³⁷

- **DIT** — есть как Depth of Inheritance (конвенция может отличаться, но смысл близок). ³⁸
- **SLOC** — частично, через `SourceLines / ExecutableLines` (это не весь набор SLOC-производных, приведённых в презентации, но базовые показатели есть). ³⁹
- **СВО/LCOM/TCC/LCC/CF и пакетные метрики** — не заявлены как часть стандартного набора Code Metrics (по крайней мере в публичной документации набора значений). ³⁷

Вывод для ВКР: встроенные средства Microsoft полезны для валидации расчёта CC/DIT/SLOC-подобных метрик и как "эталон" для совместимости, но не покрывают весь набор ООП-метрик из презентации.

Sonar-экосистема (SonarQube / Sonar analyzers)

Для cognitive complexity у Sonar есть хорошо описанные правила инкрементов (разрывы линейного чтения, вложенность, смешанные логические условия и т.д.). ¹⁹

Этот стек особенно полезен как: - источник формализованных правил cognitive complexity, - практический "бенчмарк" для сравнения собственного расчёта. ⁴⁰

Однако Sonar (в типичной конфигурации) ориентирован на метрики сложности/размера и "smell"-правила, а не на полноценный расчёт всего семейства LCOM/TCC/LCC и пакетных примитивов "out-port/in-port" именно в трактовке презентации. Это требует проверки по конкретной инсталляции и плагинам, и потому в ВКР корректно формулировать это как ограничение "не гарантирует покрытие всего набора". ¹²

NDepend

NDepend (для .NET) предоставляет богатую модель кода и метрик, включая: - DepthOfInheritance, - NbChildren, - LCOM и LCOMHS, - CyclomaticComplexity на уровне типа, и др. — это видно в API-перечне свойств сущности `IType`. ⁴¹

Также на уровне сборок/пространств имён присутствуют показатели "instability" и зависимостей (что коррелирует с пакетными метриками). ⁴²

Сопоставление с презентацией: - DIT/NOC/LCOM/CC — покрываются (хотя набор LCOM-вариантов и правила счёта методов/аксессоров нужно уточнять). ⁴³

- пакетные метрики и зависимости — доступны в терминах зависимостей сборок/неймспейсов/типов, что позволяет реализовать Ca/Ce/I/A/D и примитивы NCP/OutC/InC/HC/SPC/SCC, но точное соответствие определениям из презентации нужно устанавливать явно. ⁴⁴

Критическое замечание для ВКР: NDepend — мощный "референс-инструмент", но он не заменяет разработку собственной программы, если цель ВКР — именно разработка (и если есть ограничения по лицензированию/встраиванию).

Где "найти реализацию в интернете" и как это использовать корректно

Для будущей реализации в C# важны источники, где есть не только формула, но и воспроизводимая реализация: - исходники `Metrics.exe` доступны через `dotnet/roslyn` (Microsoft прямо описывает путь сборки и то, что инструмент поставляется как NuGet/исполняемый файл). ¹⁶

- правила вычисления СС и её роль в анализаторах описаны в Microsoft-документации (CA1502), что помогает реализовать совместимую интерпретацию. ⁴⁵

- для LCOM4 (с учётом вызовов и транзитивного графа) и примитивов package metrics существуют формальные определения в источниках по quality models и deliverables (например, SQuALE deliverable даёт определения LCOM4 и OutC/InC/HC/SPC/SCC). ²⁸

При этом ВКР должна учитывать риск "расхождения инструментов": даже при одинаковых названиях метрик инструменты могут реализовывать разные варианты, и это влияет на сравнимость результатов. ³⁵

Сравнение вариантов решения и обоснование выбора подхода для собственной программы

Для разработки программы оценки качества ООП-приложений C# (по набору метрик из презентации) реалистично рассмотреть несколько вариантов.

Вариант А: использовать готовые инструменты как "чёрный ящик"

Пример: запуск `msbuild /t:Metrics` и парсинг отчёта `Microsoft.CodeAnalysis.Metrics` либо интеграция Sonar. ⁴⁶

Плюсы: быстро, воспроизводимо для своего набора метрик, хорошо для CI. ¹⁶

Минусы: не покрывает LCOM1-LCOM5, TCC/LCC, CF и полный блок пакетных примитивов в трактовке презентации; трудно расширять "внутренности" расчёта и обеспечивать строгую спецификацию "как именно считали". ⁴⁷

Вариант В: опираться на коммерческие анализаторы как эталон

Например, использовать NDepend для валидации и/или отчётности. ⁴⁸

Плюсы: широкий набор метрик, готовые отчёты и модель кода. ⁴⁹

Минусы: зависимость от стороннего продукта (лицензирование, воспроизводимость в рамках ВКР в учебной среде, ограничение на модификацию/встраивание).

Вариант С: разработать собственный вычислитель метрик на C# на базе компиляторной инфраструктуры

Суть: использовать синтаксическое и семантическое представление кода C# (анализ AST + семантическая модель) и реализовать вычисление метрик по спецификациям из презентации (а также по первоисточникам, где они есть). ⁵⁰

Плюсы: - можно реализовать весь требуемый набор метрик и зафиксировать определения, - можно добавить режимы совместимости (например, "как Microsoft Metrics.exe" для CC/DIT или "как CK/MOOD" для СВО/LCOM/CF), - можно расширять модель на уровне пакетов (проект/namespace) и выдавать отчёты в нужном формате. ⁵¹

Минусы: - выше трудоёмкость, - нужна строгая спецификация трактовок и единое правило "что считать методом/атрибутом/связью", иначе результаты будут спорными. ⁵²

С учётом цели ВКР ("разработка программы...") и требования вычислять ООП-метрики из презентации, наиболее обоснован вариант С, при этом вариант А/В желательно использовать как: - источники проверочных примеров (валидация), - ориентиры совместимости по отдельным метрикам (CC/DIT/LCOMHS и т.п.). ⁵³

Постановка задач для дальнейшей ВКР и требования к прототипу

Ниже — формулировка задач в стиле раздела "Обзор по теме", чтобы далее перейти к методам, алгоритмам и реализации.

Сформировать **спецификацию метрик** и их формальных определений (на основе презентации и первоисточников), включая фиксацию вариантов и допущений: - DIT, NOC, NOM, WMC (с правилами учёта методов/конструкторов/аксессоров). ⁵⁴

- Cyclomatic Complexity и Cognitive Complexity (точные правила инкрементов и вложенности). ⁵⁵

- СВО, CF, MPC (что считать "связью", исключение наследования в CF). ⁵⁶

- LCOM1-LCOM5, TCC, LCC (что считать атрибутами, как учитывать вызовы методов в LCOM4, какие методы исключать). ⁵⁷

- пакетные примитивы и Martin-метрики (определить "пакет" как проект/namespace; формализовать зависимости). ⁵⁸

Разработать **модель извлечения фактов из C# кода**: - загрузка решения/проектов и построение полного списка сущностей (классы/интерфейсы/пространства имён/сборки), - получение семантических зависимостей "класс использует класс" (для СВО/CF и для пакетных метрик), - извлечение "метод использует атрибут" и "метод вызывает метод" (для LCOM/TCC/LCC).

Спроектировать **архитектуру вычислителя**: - модульное разделение “извлечение модели” → “вычисление метрик” → “генерация отчётов”, - поддержка нескольких уровней агрегации: метод → тип → namespace → проект/assembly (аналогично структуре отчёта Metrics.exe). 16

Определить **стратегию валидации**: - для CC/DIT/SLOC-подобных метрик сравниваться с Microsoft Code Metrics там, где набор совпадает. 36

- для cognitive complexity — сравниваться с правилами Sonar (и при необходимости с отчётом Sonar). 19

- для LCOM4 и package primitive metrics — проверять на синтетических примерах и на известных формульных определениях. 28

- везде фиксировать, что разнотечения возможны из-за различий в определениях, и документировать их. 35

Сформулировать требования к **выходному формату** прототипа: - отчёт по проекту/решению (например, JSON/XML), включающий значения метрик и ссылки на элементы кода, - возможность строить “срезы” по классам с наибольшими значениями риска (высокий WMC/CC, низкая связность, высокая зависимость и т.п.). 59

Наконец, в разделе ВКР необходимо заранее зафиксировать “метрический прагматизм”: метрики полезны для сравнения и мониторинга, но сами по себе не заменяют инженерного анализа и могут стимулировать нецелевую оптимизацию. Это прямо отмечено в заключении презентации.

3

1 2 3 4 5 6 7 8 9 10 11 15 22 25 53 55 56 57 59 cs.petrsu.ru

<https://cs.petrsu.ru/~kulakov/courses/develop/lectures/06.metrics.pdf>

12 35 <https://arxiv.org/abs/2103.04506>

<https://arxiv.org/abs/2103.04506>

13 https://www.tusharma.in/thesis/PhD_thesis_TusharSharma.pdf

https://www.tusharma.in/thesis/PhD_thesis_TusharSharma.pdf

14 21 26 27 <https://www.eso.org/~tcsmgr/oowg-forum/TechMeetings/Articles/OOMetrics.pdf>

<https://www.eso.org/~tcsmgr/oowg-forum/TechMeetings/Articles/OOMetrics.pdf>

16 34 36 39 46 50 51 52 <https://learn.microsoft.com/en-us/visualstudio/code-quality/how-to-generate-code-metrics-data?view=visualstudio>

<https://learn.microsoft.com/en-us/visualstudio/code-quality/how-to-generate-code-metrics-data?view=visualstudio>

17 54 <https://www.mccabe.com/pdf/mccabe-nist235r.pdf>

<https://www.mccabe.com/pdf/mccabe-nist235r.pdf>

18 45 <https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca1502>

<https://learn.microsoft.com/en-us/dotnet/fundamentals/code-analysis/quality-rules/ca1502>

19 20 40 <https://rules.sonarsource.com/javascript/tag/brain-overload/rspec-3776/>

<https://rules.sonarsource.com/javascript/tag/brain-overload/rspec-3776/>

23 24 <https://scispace.com/pdf/an-overview-of-object-oriented-design-metrics-2br8ar4xyi.pdf>

<https://scispace.com/pdf/an-overview-of-object-oriented-design-metrics-2br8ar4xyi.pdf>

28 32 58 squale.org

https://www.squale.org/quality-models-site/research-deliverables/WP1.1_Software-metrics-for-Java-and-Cpp-practices_v2.pdf

²⁹ <https://gupea.ub.gu.se/server/api/core/bitstreams/49bbeda6-2cbd-4637-9888-5e066fd3db80/content>

<https://gupea.ub.gu.se/server/api/core/bitstreams/49bbeda6-2cbd-4637-9888-5e066fd3db80/content>

³⁰ ³¹ <https://www.cs.colostate.edu/~bieman/Pubs/aowsm95.pdf>

<https://www.cs.colostate.edu/~bieman/Pubs/aowsm95.pdf>

³³ ⁴⁴ <https://linux.ime.usp.br/~joaomm/mac499/arquivos/referencias/oodmetrics.pdf>

<https://linux.ime.usp.br/~joaomm/mac499/arquivos/referencias/oodmetrics.pdf>

³⁷ ³⁸ ⁴⁷ **Code metrics values - Visual Studio**

https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values?view=visualstudio&utm_source=chatgpt.com

⁴¹ **IType Interface Properties**

https://www.ndepend.com/API/NDepend.API~NDepend.CodeModel.IType_properties.html?utm_source=chatgpt.com

⁴² **IAssembly Interface Properties**

https://www.ndepend.com/API/NDepend.API~NDepend.CodeModel.IAssembly_properties.html?utm_source=chatgpt.com

⁴³ ⁴⁸ ⁴⁹ **IType Interface Properties**

https://www.ndepend.com/API/NDepend.API~NDepend.CodeModel.IType_properties.html