

DISTANCE OF COUPLING КАК МЕТРИКА КАЧЕСТВА ПРОГРАММНОГО КОДА КРУПНОМАСШТАБНЫХ СИСТЕМ

Селютин А.Д., Богомолов А.С.

Саратовский научный центр РАН, Саратов, Россия

aseliutin@ya99.ru, alexbogomolov@yandex.ru

Фурсов С.В.

Институт проблем управления им. В.А. Трапезникова РАН, Москва, Россия

fursovs@ipu.ru

Аннотация. В работе представлена методика автоматизированного расчета метрики качества Distance Of Coupling, отражающая глубину вычислительной связности переменных в программной системе. Проведен анализ применимости метрики к крупномасштабным информационным системам и обоснована ее архитектурная информативность.

Ключевые слова: distance of coupling, качество кода, связность кода, сцепленность кода, метрики кода, программная инженерия.

Введение

Вопрос оценки архитектурного качества программного обеспечения остается одной из ключевых задач в программной инженерии. Несмотря на наличие множества устоявшихся метрик, таких как связность (cohesion) [1], сцепленность (coupling) [2], глубина дерева наследования (depth of inheritance) [3] сохраняется потребность в новых, более точных и интуитивно понятных способах количественной оценки структурной организации кода.

В работе [4] предложена оригинальная метрика под названием Distance Of Coupling (DoC). В отличие от традиционных метрик, фокусирующихся на прямых зависимостях между компонентами системы, DoC измеряет глубину опосредованного влияния одного объекта на другой через цепочку промежуточных преобразований. Эта метрика отражает, насколько далеко во внутренней логике системы распространяется воздействие одного элемента на другие по пути вычислений и передачи данных.

Формально, пусть переменная x влияет на переменную y через цепочку промежуточных вычислений x_1, x_2, \dots, x_n , тогда Distance Of Coupling между x и y можно определить как:

$$DoC(x, y) = \min\{n \in \mathbb{N} \mid x \rightarrow x_1 \rightarrow \dots \rightarrow x_n = y\}, \quad (1)$$

где $n \in \mathbb{N}$ – минимальное количество шагов преобразования, необходимое для получения y из x , а оператор \rightarrow – непосредственное вычислительное влияние (значение переменной справа зависит от значения переменной слева).

Метрика Distance Of Coupling позволяет выявить скрытые, неочевидные зависимости, которые могут значительно затруднять сопровождение, тестирование и рефакторинг кода [5]. Чем выше значение DoC между двумя элементами системы, тем сильнее их логическая связанность, несмотря на потенциальное отсутствие прямого взаимодействия.

Несмотря на концептуальную простоту и практическую значимость, метрика DoC остается малоизученной в научной литературе.

Цель исследования – оценить применимость DoC в качестве инструмента архитектурного анализа и выявить типовые паттерны сильной логической связности в реальных системах. Для этого в данной работе предлагается формализация метрики, алгоритм сбора статистических данных, а также проводится эмпирическое исследование на множестве Java-проектов с открытым исходным кодом.

1. Постановка задачи

Пусть программа P представлена в виде множества выражений и операторов, действующих над переменными. Обозначим через V множество всех переменных, порождаемых в ходе исполнения P , и введем ориентированный граф вычислительных зависимостей:

$$G = (V, E), \quad (2)$$

где вершины V соответствуют переменным программы, а каждое ориентированное ребро $(u, v) \in E$ означает, что значение переменной v прямо зависит от значения переменной u (например, через присваивание, метод, выражение или индексный доступ).

Определим метрику Distance Of Coupling между двумя переменными $x, y \in V$, если существует путь от x к y , как длину самого длинного такого пути:

$$DoC(x, y) = \max\{\ell(p) \mid p = (x, u_1, \dots, u_k = y), (u_i, u_{i+1}) \in E\}, \quad (3)$$

где $\ell(p) = k$ – длина пути в количестве промежуточных шагов.

Если путь не существует, считаем $DoC(x, y) = \infty$ (или значение не определено).

Таким образом, вычисление DoC сводится к поиску длиннейших путей во взвешенном графе (с единичными весами ребер) между всеми парами переменных.

Полученное множество значений DoC позволит судить о степени связности компонентов программного кода и выявлять потенциально проблемные участки архитектуры.

2. Применимость метрики к крупномасштабным системам

Современные программные системы, особенно в корпоративной и распределенной среде, характеризуются высокой степенью структурной и логической сложности. По мере роста числа компонентов, слоев абстракции и внешних зависимостей, усиливается риск появления неявных логических связей между удаленными частями кода. Это приводит к скрытому зацеплению, при котором изменение в одном фрагменте программы может непредсказуемо отразиться на других.

В таких условиях традиционные метрики, например, LCOM (Lack of Cohesion in Methods) [6], TCC (Tight Class Cohesion) и LCC (Loose Class Cohesion) [7] зачастую оказываются недостаточными для выявления глубинных причин сложности сопровождения. В частности:

В работе [8] анализируется метрика Coupling Between Objects (CBO), измеряющая количество классов, с которыми данный класс связан напрямую. Авторы подчеркивают ее полезность при выявлении чрезмерно связанных компонентов, однако CBO игнорирует глубину трансформационного влияния: она фиксирует лишь факт связи, но не учитывает, насколько далеко эта связь распространяется по логике выполнения. Таким образом, CBO остается поверхностной метрикой, неспособной отражать опосредованное зацепление.

В исследовании [9] рассматривается структурная метрика Data Flow Complexity, основанная на подсчете дуг в графе потока данных. Она применима на уровне отдельных процедур и позволяет судить о локальной сложности, но плохо масштабируется на уровень межклассовых взаимодействий в больших системах. Кроме того, она не предоставляет меры расстояния между источником и приемником данных, что ограничивает ее аналитическую ценность при анализе логических зависимостей между частями кода.

В свою очередь метрика Distance Of Coupling (DoC) дополняет существующий инструментарий, позволяя количественно оценить глубину опосредованного влияния между переменными или объектами в программе. В отличие от поверхностных метрик, DoC отражает, насколько далеко распространяется влияние значения переменной по логике выполнения. Таким образом, использование метрики DoC при анализе программных систем дает возможность:

- Выявлять участки кода с потенциально опасной логической связанностью.
- Локализовать архитектурные сбойные зоны, где каскад изменений наиболее вероятен.
- Приоритизировать усилия по рефакторингу на основе объективных численных оценок.
- Строить агрегированные профили логической сцепленности на уровне классов, пакетов и модулей программной системы.

Особое значение исследование метрики DoC приобретает в контексте языка Java, который на протяжении десятилетий остается доминирующей платформой в разработке крупномасштабных систем, в частности, в банковском ПО, телекоммуникациях, страховании и госсекторе [10]. Язык Java отличается жесткой типизацией, строго определенной объектной моделью и развитым инструментарием статического анализа, что делает его особенно подходящим для построения точных графов вычислительных зависимостей [11].

В связи с этим в дальнейшем исследование будет сосредоточено именно на анализе проектов с открытым исходным кодом на Java, позволяющих масштабировать протестировать поведение и интерпретируемость метрики DoC в условиях промышленной архитектурной сложности.

3. Алгоритм сбора данных

Для оценки значений метрики Distance Of Coupling (DoC) в масштабах реальных Java-проектов был разработан алгоритм автоматизированного анализа, состоящий из следующих этапов:

1. *Построение графа вычислительных зависимостей.* Каждый анализируемый проект представляется в виде ориентированного графа $G = (V, E)$, где V – множество переменных (в том числе временных, локальных и полей объектов), участвующих в вычислениях; $E \subset V \times V$ – множество дуг, отражающих факт использования значения одной переменной при формировании другой. Ребра $(v_i, v_j) \in E$ добавляются в граф, если значение переменной v_j явно или неявно зависит от v_i , например, через арифметические операции, вызовы методов, преобразования или разбиение строк. Анализ производится на уровне байткода или AST, с возможностью восстановления цепочек трансформаций;
2. *Вычисление значений DoC.* Для каждой переменной $v \in V$ определяется множество достижимых из нее переменных $R_v \subset V$, а также максимальные расстояния до них. Значение Distance Of Coupling для переменной v определяется как:

$$DoC(x, y) = \max_{r \in R_v} dist(v, r), \quad (4)$$

где $dist(v, r)$ – длина самого длинного пути от v до r в графе G , измеряемая в количестве ребер. Таким образом, метрика фиксирует максимальную логическую глубину влияния, оказываемого значением переменной на вычисления в программе;

3. *Фильтрация и нормализация.* Из анализа исключаются: переменные, находящиеся в пределах методов-оберток (геттеры/сеттеры [12]); переменные, не участвующие в трансформациях; автогенерируемый код (например, Lombok [13]), чтобы избежать искажения статистики. Для возможности сравнения между проектами разной размерности применяется нормализация: значения агрегатов DoC сопоставляются с числом классов, методов или строк кода в проекте (например, медиана DoC на 1000 строк кода);
4. *Агрегация.* Поскольку в реальных проектах количество переменных велико, то отдельные значения DoC неинтерпретируемы без статистической обработки. Для обобщения результатов по проекту используются следующие агрегированные показатели:
 - *Медиана DoC.* Устойчива к выбросам, отражает глубину сцепленности кода.
 - *Максимальное значение DoC.* Указывает на наиболее глубоко встроенную в логику переменную.
 - *90-й перцентиль.* Характеризует верхнюю границу распространенных значений DoC.
 - *Дисперсия DoC.* Отражает неоднородность архитектурной глубины.

На основе вышеописанных шагов на рис. 1 приведена блок-схема алгоритма сбора данных по метрике DoC.

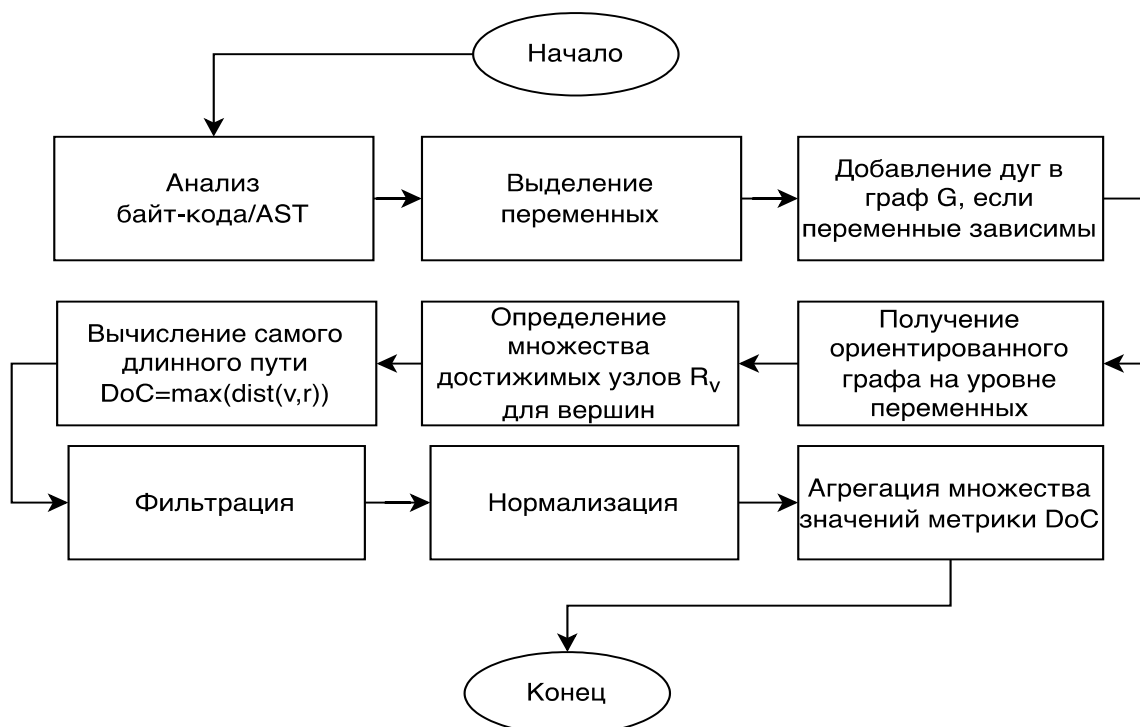


Рис. 1. Блок-схема сбора данных по метрике Distance Of Coupling (DoC)

4. Модельный пример

Для верификации предложенной методики и анализа поведения метрики Distance Of Coupling (DoC) в условиях реальных программных систем было проведено экспериментальное исследование на совокупности Java-проектов с открытым исходным кодом. В выборку вошло 157 репозитория, отобранных из открытых источников (GitHub [14]) с различными характеристиками по количеству строк кода (LOC (Lines Of Code)) [15], времени создания и архитектурной платформе (Java EE, Spring Boot [16]).

Отбор проектов производился с целью охвата широкого спектра: от малых утилитарных библиотек до масштабных корпоративных систем. Для всех объектов исследования была проведена предварительная очистка данных, включающая исключение автогенерируемого кода, методов-оберток и нефункциональных компонентов. Алгоритм, описанный ранее, применялся к каждому проекту с последующим вычислением агрегированных статистик: медиана DoC, 90-й перцентиль, максимум и дисперсия. В таблице 1 приведены обобщенные результаты значений метрики DoC.

Таблица 1. Обобщенные результаты

Показатель	Среднее значение
Медиана DoC	3,2
90-й перцентиль DoC	7,8
Максимум DoC	15,4
Дисперсия DoC	4,1
Среднее количество переменных	12300
Среднее количество классов	670

4.1. Категориальный анализ

В рамках исследования был проведен категориальный анализ полученных значений метрики Distance Of Coupling. Такой подход обусловлен необходимостью установить связь между характеристиками программных систем и глубиной вычислительной связности переменных. Анализ отражает специфику предметной области Java-разработки, где существенную роль играют как масштаб системы, так и архитектурные и технологические решения.

В частности, были выделены следующие категории, отражающие распространенные отличительные признаки программных проектов:

- Размер системы в строках кода (LOC).
- Используемая архитектурная платформа. В частности, Spring Boot или Java EE.
- Период начала разработки.

Каждая категория была рассмотрена отдельно с целью выявления типовых закономерностей и потенциального архитектурного смысла высоких или низких значений DoC.

В таблице 2 представлены показатели метрики DoC для категории масштабности Java-проекта. Для оценки влияния масштабности на значения DoC проекты были разбиты по числу строк исходного кода.

Таблица 2. Влияние масштабности проекта

Категория	Средняя медиана DoC	Средний максимум DoC
Малые (<50 тыс. LOC)	2,6	10,1
Средние (50–200 тыс. LOC)	3,4	13,8
Крупные (>200 тыс. LOC)	4,1	18,2

Наблюдается устойчивая тенденция к увеличению глубины сцепленности (DoC) в крупных системах. Это подтверждает гипотезу о росте архитектурной сложности с масштабом кода.

Далее было проведено сравнение по платформам Spring Boot и Java EE по группам из 30 проектов каждой категории, сопоставимых по размеру. Результаты представлены в таблице 3.

Таблица 3. Сравнение архитектурных платформ

Категория	Медиана DoC	90-й перцентиль	Дисперсия
Spring Boot	2,6	6,4	3,2
Java EE	3,4	9,3	4,7

Программные системы на базе Spring Boot демонстрируют более локализованное распространение зависимостей между переменными. Это может быть связано с преобладанием декларативного конфигурирования и шаблонных компонентов.

Для анализа эволюции архитектурной связности в программных системах выборка была разбита по году начала разработки на четыре интервала: до 2010 года, с 2010 по 2019 годы, с 2020 по 2023 годы, и последние проекты, начатые в 2024–2025 годах. В таблице 4 представлены усредненные значения по каждой группе.

Таблица 4. Влияние времени разработки

Категория	Медиана DoC	Максимум DoC	Дисперсия
До 2010 г.	3,8	17,6	4,9
2010-2019 гг.	3,4	15,2	4,3
2020-2023 гг.	2,9	11,8	3,2
2024-2025 гг.	2,5	9,4	2,7

Наблюдается отчетливая тенденция к снижению значений метрики DoC в более новых проектах. Это может быть обусловлено внедрением современных архитектурных подходов, таких как микросервисные и функциональные модели, а также широким использованием библиотек, абстрагирующих вычислительные зависимости. Результаты подтверждают гипотезу о возрастающей структурной упрощенности современных программных систем.

5. Заключение

В работе была рассмотрена метрика Distance Of Coupling (DoC), предложенная в контексте оценки архитектурной сложности программных систем. В отличие от традиционных метрик сцепления и связанности, DoC фокусируется на глубинной вычислительной зависимости между переменными в рамках программного исполнения. Подобный подход позволяет зафиксировать не только наличие связи, но и степень ее транзитивного влияния в логике системы, что делает метрику особенно ценной для анализа структурных и архитектурных характеристик кода.

Разработанная методика автоматизированного анализа, основанная на построении графа вычислительных зависимостей и вычислении на его основе значений DoC, показала свою применимость на практике. Исследование, охватившее 157 Java-проектов различного масштаба, архитектурной направленности и периода разработки, позволило выявить ряд закономерностей, подтверждающих теоретические гипотезы. Установлено, что глубина сцепления (DoC) увеличивается с ростом масштаба программной системы, выше в проектах на платформе Java EE по сравнению со Spring Boot, а также имеет тенденцию к снижению в более современных проектах, что может свидетельствовать о постепенном упрощении логики программных компонентов.

Полученные результаты подтверждают архитектурную чувствительность метрики DoC и ее потенциал в качестве инструмента статического анализа качества программного кода. Особенно значимо, что агрегированные характеристики DoC (медиана, перцентили, дисперсия) позволяют объективно оценивать структуру системы на уровне всего проекта, не ограничиваясь анализом отдельных модулей. Это делает DoC перспективным дополнением к существующему арсеналу метрик, применяемых при автоматическом контроле качества, сопровождении и рефакторинге программных систем.

Следует также отметить, что использование DoC не исключает других показателей качества, а напротив, дополняет их, предлагая новый взгляд на внутреннюю архитектурную логику программы. В перспективе возможно дальнейшее развитие методики, включая поддержку других языков программирования, улучшение точности анализа байткода и интеграцию с существующими средствами статического анализа.

Таким образом, предложенная метрика Distance Of Coupling и основанный на ней подход обладают высокой степенью обобщаемости, инструментальной реализуемостью и могут быть полезны как для исследовательского сообщества, так и для практикующих инженеров, заинтересованных в повышении структурной устойчивости и поддерживаемости программных систем.

Литература

1. Cohesion и Coupling: отличия. <https://habr.com/ru/articles/568216/> (дата обращения: 30.05.2025).
2. Tsui F., Karam O. An Empirical Study on Software Test-Case Development Complexity and Software Code Cohesion // School of Computing and Software Engineering Southern Polytechnic State University. – 2008. – P. 1–10.
3. Code metrics - Depth of inheritance (DIT). <https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metrics-depth-of-inheritance?view=vs-2022> (дата обращения: 03.06.2025).

4. New Metric: The Distance of Coupling. <https://www.yegor256.com/2020/10/27/distance-of-coupling.html> (дата обращения: 25.05.2025).
5. Кушников В.А., Богомолов А.С., Селютин А.Д. Количественный анализ качества ERP-систем // Естественные и технические науки. – 2024. – № 4(191). – С. 166–170.
6. Chidamber S.R., Kemerer C.F. A metric suite for object oriented design // IEEE Trans. Softw. Eng. – 1994. – Vol. 20, № 6. – P. 476–493.
7. Bieman J.M., Kang B. Cohesion and reuse in an object-oriented system // ACM SIGSOFT Symposium on Software Reusability. – Colorado, 1995.
8. Shatnawi R.A. Quantitative Investigation of the Acceptable Risk Levels of Object-Oriented Metrics in Open-Source Systems // IEEE Transactions on Software Engineering. – 2010. – Vol. 36, № 2. – P. 216–225.
9. Karkare B., Khedker U. Complexity of Data Flow Analysis for Non-Separable Frameworks // International Workshop on Program Analysis and Compilation. – 2006. – P. 1–13.
10. Резчиков А.Ф., Кушников В.А., Твердохлебов В.А. Управление и развитие крупномасштабной системы // Управление развитием крупномасштабных систем MLSD'2017: Материалы Десятой международной конференции: в 2-х томах, Москва, 02–04 октября 2017 года / Институт проблем управления им. Трапезникова В.А.; Российская академия наук; под общей редакцией Васильева С.Н., Цвиркуна А.Д. Том I. – Москва: Институт проблем управления им. Трапезникова В.А. РАН, 2017. – С. 107–119.
11. Кушников В.А., Резчиков А.Ф., Иващенко В.А., Богомолов А.С., Филимонок Л.Ю. Математическое моделирование процессов функционирования сложных систем // Математические методы в технике и технологиях – ММТТ. – 2017. – Т. 11. – С. 83–91.
12. Резчиков А.Ф., Кушникова Е.В., Кушников О.В., Богомолов А.С. Постановка задачи управления целенаправленным процессом проектирования сложных систем // Управление развитием крупномасштабных систем (MLSD'2023): Труды Шестнадцатой международной конференции, Москва, 26–28 сентября 2023 года. – Москва: Институт проблем управления им. Трапезникова В.А. РАН, 2023. – С. 55–62.
13. Project Lombok. <https://projectlombok.org/> (дата обращения: 03.06.2025).
14. GitHub. <https://github.com/> (дата обращения: 03.06.2025).
15. Boehm B., Abts C., Brown A.W., Chulani S., Clark B.K., Horowitz E., Madachy R., Reifer D.J., Steece B. Software Cost Estimation with COCOMO II // Upper Saddle River, NJ: Prentice Hall, 2000. – P. 544.
16. Бровко А.В., Ермаков А.В. Разработка Java приложений: Учебное пособие. В 2 частях. Часть 1 // Саратов: Саратовский государственный технический университет имени Гагарина Ю.А., 2015. – 136 с.