DOI: 10.15514/ISPRAS-2023-35(6)-3

Оценки сложности программного обеспечения на основе косвенных связей

[1] *Х. Навас-Су*, ORCID: 0000-0003-3431-0122 <*jnavas@tec.ac.cr*>
[2] *А. Гонсалес-Торрес*, ORCID: 0000-0001-5427-0637 <*antonio.gonzalez@tec.ac.cr*>

[1] *Коста-Риканский технологический институт, компьютерная школа, Коста-Рика.*
[2] *Коста-Риканский технологический институт, факультет вычислительной техники, Коста-Рика.*

Аннотация. Разработка программного обеспечения может быть длительным и дорогостоящим процессом, требующим значительных усилий. Перед разработчиками часто ставят задачи по программированию или внесению изменений в существующие программы так, чтобы общая сложность не увеличилась. Понятно, что прежде, чем вносить какие-либо изменения, важно понять зависимости между компонентами программы. Однако по мере роста размеров программ менеджерам проектов становится все сложнее обнаруживать косвенные связи между компонентами. Эти скрытые связи могут усложнять систему, приводить к неточной оценке необходимых затрат и ставить под угрозу качество получающихся программ. Чтобы решить эти проблемы, данное исследование направлено на выработку набора мер, которые дополняют теорию измерений и выявляют скрытые связи между компонентами программного обеспечения, расширяя сферу применения, увеличивая эффективность и полезность общепризнанных метрик программного обеспечения. Исследование велось в двух главных направлениях: (1) как измерения косвенных зависимостей могут помочь разработчикам при сопровождении программ и (2) как метрики косвенных связей могут количественно оценивать сложность и размер программного обеспечения, используя взвешенные различия между методами. В исследовании представлен комплекс мер, призванных помочь менеджерам проектов и разработчикам в управлении проектами и их сопровождении. Используя возможности измерений косвенных связей, эти меры могут повысить качество и эффективность процессов разработки и поддержки программного обеспечения.

Ключевые слова: Косвенное связывание; сопровождение программного обеспечения; удобство сопровождения; метрики.

Для цитирования: Навас-Су Х., Гонсалес-Торрес А. Оценки сложности программного обеспечения на основе косвенных связей. Труды ИСП РАН, том 35, вып. 6, 2023 г., стр. 43–74. DOI: 10.15514/ISPRAS–2023–35(6)–3.

Measuring Software Complexity using Indirect Coupling

[1] *J. Navas-Su*, ORCID: 0000-0003-3431-0122 <*jnavas@tec.ac.cr*>
[2] *A. Gonzalez-Torres*, ORCID: 0000-0001-5427-0637 <*antonio.gonzalez@tec.ac.cr*>

[1] *Computing School, Costa Rica Institute of Technology, Costa Rica*.
[2] *Computer Engineering Department, Costa Rica Institute of Technology, Costa Rica.*

Abstract. Software development can be a time-consuming and costly process that requires a significant amount of effort. Developers are often tasked with completing programming tasks or making modifications to existing code without increasing overall complexity. It is essential for them to understand the dependencies between the program components before implementing any changes. However, as code evolves, it becomes increasingly

challenging for project managers to detect indirect coupling links between components. These hidden links can complicate the system, cause inaccurate effort estimates, and compromise the quality of the code. To address these challenges, this study aims to provide a set of measures that leverage measurement theory and hidden links between software components to expand the scope, effectiveness, and utility of accepted software metrics. The research focuses on two primary topics: (1) how indirect coupling measurements can aid developers with maintenance tasks and (2) how indirect coupling metrics can quantify software complexity and size, leveraging weighted differences across techniques. The study presents a comprehensive set of measures designed to assist developers and project managers with project management and maintenance activities. Using the power of indirect coupling measurements, these measures can enhance the quality and efficiency of software development and maintenance processes.

Keywords: Indirect coupling**;** maintainability**;** software maintenance**;** metrics**.**

For citation: Navas-Su J., Gonzalez-Torres A. Measuring Software Complexity using Indirect Coupling. *Trudy ISP RAN/Proc. ISP RAS*, vol. 35, issue 6, 2023. pp. 43-74 (in Russian). DOI: 10.15514/ISPRAS-2023-35(6)-3.

1. Введение

Сопровождение программного обеспечения включает в себя внесение в него адаптивных, совершенствующих, корректирующих и профилактических изменений. По мере развития программного обеспечения его сложность и качество могут улучшаться или ухудшаться [1]. Таким образом, основными целями сопровождения программного обеспечения являются продление срока его службы и сохранение его целостности с течением времени [2].

Управление развитием системы – это задача, которую должны решить менеджеры и программисты [3]. Для изучения модификаций и их влияния на программное обеспечение требуются эффективные методы, такие как идентификация косвенных связей и измерение.

Отслеживать изменения и измерять их влияние позволяют графы косвенных связей [4]. Для отслеживания изменений в зависимостях программных компонентов друг от друга программистам необходим структурный подход, каким, например, является использование ориентированных графов. Для численной оценки вклада одного программного компонента в другие требуется иметь представление о всей системе в целом, которое поможет приписать элементам графа весовые значения. Кроме того, чтобы лучше понять влияние изменений, вносимых в программы, на ее характеристики, разработчики должны иметь возможность сравнивать метрические показатели до и после внесения изменений.

Построенный граф позволяет отслеживать эффекты, вызванные изменениями одного компонента, но проявляющиеся в других компонентах, а также измерять потенциальное влияние планируемой модификации. Этот метод отслеживает связи между компонентами, которые выстраиваются в цепочки, и анализирует эти связи. Таким образом, в результате можно получить представление о внутренних зависимостях между применяемыми в программах методами, а также о том, как эти методы совместно выполняют те или иные функции системы.

Помощь при построении вышеупомянутой структуры может оказать обнаружение явных (прямых, непосредственных) связей (Direct Coupling, DC) [5-10], которые могут существовать между сущностями программ, такими как модули, классы, объекты, методы или элементы данных. Явные связи обычно классифицируются по той силе, с которой элементы соединены между собой [11, 12]. Комбинирование формаций явных связей создает конфигурацию сети косвенных связей (Indirect Coupling, IC), упрощая анализ и способствуя пониманию влияния изменений на другие компоненты системы [13-16]. Прямые связи транзитивно замыкаются на косвенных связях.

Следующие разделы настоящей статьи построены таким образом: в разделе 2 представлена постановка исследовательской задачи, в разделе 3 объясняются некоторые основы теории измерений. Далее в разделе 4 представлены расчетные критерии метрик, в разделе 5 описаны методы сбора данных, в разделе 6 представлены результаты, а в разделе 7 обсуждаются основные выводы.

2. Постановка задачи

Изучение косвенных связей (IC) включает в себя определение концепций [17, 18], теоретических основ [8], методов графов зависимостей [19–21], метрик [22–27], методов извлечения и обнаружения графов неявных зависимостей (косвенных связей) [12, 28] и создание аналитических инструментов [29–31]. Однако многие исследовательские подходы сосредоточены исключительно на идентификации графов косвенных связей и расчете метрических показателей на основе подсчета входящих и исходящих парных связей, передачи параметров и ссылок на элементы данных. Эти подходы не учитывают различия в степени вклада компонентов в исследуемый элемент.

Для восполнения этого пробела, необходимо учитывать такие понятия, как хрупкость и жесткость (иначе – рискованность), которые напрямую связаны с косвенными связями [32]. Хрупкость означает зависимость объекта от множества других компонентов, что делает его уязвимым для изменений, вызванных артефактами, от которых он зависит. Напротив, жесткость (или рискованность) отражает влияние модификаций объекта на зависимые от него артефакты. Таким образом, хрупкость элементов растет с увеличением количества влияющих на них [32], а их жесткость растет с увеличением количества зависимых от них артефактов.

В этой статье предлагается метод обнаружения графов косвенных связей (indirect coupling graph, ICG), а также набор метрик для измерения вклада в функциональные возможности программного обеспечения. Графы косвенных связей – это направленные ациклические графы (directed acyclic graph, DAG), которые фиксируют зависимости между методами, используемыми в программе в конкретной ситуации. Вес графа косвенных связей равен весу его корневого узла, вычисляемому рекурсивно путем суммирования весов всех узлов, ведущих от корня к листьям графа. Аналогичным образом, вес некоторого узла в этом графе рекурсивно рассчитывается как сумма весов достигаемых из него элементов. Следовательно, сложность метода прямо пропорциональна количеству факторов, как прямых, так и косвенных, от которых он зависит, а метод с многочисленными уровнями зависимостей, вероятно, будет более сложным, чем метод с меньшим количеством зависимостей.

Итак, это исследование предлагает подход к построению графов косвенной связи на уровне методов и определяет набор метрик для измерения сложности и размера элементов, основанных на концепциях жесткости и хрупкости [32]. Подход позволяет анализировать распространение изменений и ошибок, учитывать влияние компонентов на их партнеров по графу косвенной связи. В этом подходе для выявления и измерения косвенной связи выбирается степень детализации на уровне методов классов, поскольку в объектно-ориентированном программировании именно методы классов являются базовыми функциональными единицами. Методы – это сущности, на основе которых программисты, организуя их правильное взаимодействие, создают сложную функциональность.

Метрики программного обеспечения играют решающую роль в разработке программного обеспечения; их разработка требует высокого уровня теоретической и математической строгости. Достоверность результатов, описанных в данной статье, подтверждается той теоретической основой, которая положена в основу предлагаемого набора метрик, а также использованием общепризнанных стандартов [7, 33]. Формальная теория измерений применяется для определения набора мер на уровне детализации метода, основанного на концепциях устойчивости и неустойчивости косвенной связи [34].

В исследовании с некоторыми модификациями используются теоретический подход и обозначения, предложенные в работе [7], а также задействованы критерии проверки метрик для оценки и объяснения требуемых качеств (см. раздел 4). Все предлагаемые метрики вводятся в разделе 3, где уделяется особое внимание их формальным определениям, теоретическим основам и аналитическим оценкам.

Для демонстрации свойств и приложений метрик в этой статье использованы экспериментальные данные, полученные из коммерческих проектов и проектов с открытым исходным кодом. В разделе 5 описана методика сбора эмпирических данных, алгоритм и программный инструментарий, разработанный специально для этой цели, там же представлен набор данных используемых программных систем. Кроме того, в разделе 7 по каждой метрике даны основанные на собранных статистических данных рекомендации, предназначенные для руководителей проектов и разработчиков.

В статье тщательно и строго представлен подход к разработке метрик программного обеспечения с упором на теоретические основы, критерии проверки и экспериментальные данные. Предлагаемые метрики могут помочь оценить качество программного обеспечения, выявить потенциальные риски и уязвимости и принять обоснованные решения в проектах разработки программного обеспечения. Таким образом, данное исследование затрагивает следующие исследовательские вопросы.

RQ1. Как измерить сложность и размер программного обеспечения с помощью косвенных связей, чтобы воспользоваться преимуществами взвешенных различий между методами?

RQ2. Как метрики косвенной связи могут помочь программистам выполнять задачи по сопровождению программного обеспечения?

3. Теоретическая основа метрики косвенных связей

Пусть $S \in \mathbb{S}$ – множество всех программных систем, и пусть $S \in \mathbb{S}$ – некоторая программная система. Система S имеет ациклический граф потока управления $G = (V, A)$. Этот граф описывает поток управления между каждой парой методов в S, причем поток управления здесь означает передачу сообщений или обращения к методу. Следовательно, $V = \{v_1, v_2, v_3, \ldots\}$ – это конечное множество, содержащее все методы, объявленные в исходном коде S, а $A = \{a_1, a_2, a_3, \ldots\}$ – это конечное множество, содержащее информацию о потоке управления S. Каждая дуга $a_k \in A$ представляет собой обращение одного из методов исходного кода S к другому, например $a_k = (v_i, v_j)$, где $v_i, v_j \in V$, $v_i \neq v_j$ и $v_i \rightarrow v_j$.

Каждую систему S можно обобщенно представлять как эмпирическую систему отношений E [7]. Формальное определение этой эмпирической реляционной системы представляет собой:

$$E \equiv (V, T, ER, EO) \quad (1)$$

Первый компонент $V = \{v_1, v_2, v_3, \ldots\}$ из E в (1) – это то же самое множество V всех методов системы S.

Второй элемент $T = \{t_1, t_2, t_3, \ldots\}$ представляет собой множество транзитивных замыканий [35, p. 353], каждое t_i есть транзитивное замыкание подграфа косвенных связей ICG_i для соответствующего метода v_i, где $1 \leq i \leq |V|$. Мощность этого множества равна мощности V, то есть $|V| = |T|$. Транзитивные замыкания в T могут быть *прямыми транзитивными замыканиями* или *обратными транзитивными замыканиями*, в зависимости от направления, в котором рассматривается транзитивность. В первом случае прямой подграф $ICG_i^f = \left(V_i^f, A_i^f\right)$ содержит каждый простой путь из G, начинающийся с v_i. Согласно этому определению, $t_i^f = ICG_i^{f*} = \left(V_i^f, A_i^{f*}\right)$ это такой граф, что $V_i^f \subseteq V$, и A_i^{f*} содержит ребро (v_j, v_k), если и только если в ICG_i^f существует простой путь от v_j к v_k. В другом случае обратный подграф $ICG_i^r = (V_i^r, A_i^r)$ содержит все простые пути из G, заканчивающиеся в v_i. По этому определению, $t_i^r = ICG_i^{r*} = (V_i^r, A_i^{r*})$ это такой граф, что $V_i^r \subseteq V$, и A_i^{r*} содержит ребро (v_j, v_k), если и только если в ICG_i^r существует простой путь от v_j к v_k.

В парадигме объектно-ориентированного программирования мы управляем сложностью программных систем, используя модульность, инкапсуляцию, сокрытие информации и механизмы повторного использования кода. Строительными блоками этой парадигмы

являются абстракции реальных объектов, состояние и поведение которых моделируются с помощью атрибутов и методов. Объекты в этой парадигме взаимодействуют посредством передачи сообщений, так что, когда метод v_i отправляет сообщение другому методу v_j, он запускает процесс обмена сообщениями, который следует несколькими путями, достигая некоторых или всех методов в ICG_j.

Третий элемент тождества (1) $ER = \{er_1, er_2, er_3, \ldots\}$ представляет собой конечное множество действительных эмпирических отношений между каждой парой методов из V. К таким эмпирическим отношениям, сравнивающим размер или сложность двух методов относятся следующие:

- *больше,*
- *меньше,*
- *такой же большой, как,*
- *сложнее,*
- *менее сложный,*
- *такой же сложный, как.*

Например, если количество строк кода метода v_i меньше количества строк кода метода v_j, тогда v_i меньше, чем v_j в пересчете на строки кода, аналогично, если число цикломатической сложности метода v_j больше, чем число цикломатической сложности метода v_j, тогда v_j сложнее, чем v_j с точки зрения цикломатической сложности.

Наконец, последний компонент тождества (1) $EO = \{eo_1, eo_2, eo_3, \ldots\}$ представляет собой набор допустимых эмпирических бинарных операций над любой парой методов из V. Над парой методов v_k и v_l разрешены такие операции, как операции *добавить вызов* и *удалить вызов* от v_k к v_l. Предположим, мы выбираем методы $v_k, v_l \in t_i$ и выполняем операцию *удалить вызов* от v_k к v_l. С одной стороны, если удаленное ребро не отсоединяет ни один подграф от ICG_i, тогда t_i остается неизменным, но с другой стороны, если удаленное ребро изолирует подграф от подграфа зависимостей, то t_i меняется, поскольку теперь у элемента меньше методов и ребер. Аналогичный эффект возникает при добавлении вызова, но на этот раз либо t_i не меняется, либо на графе появляются новые методы и новые ребра.

Введем над множеством $\mathbb{S}$ бинарное отношение $\triangleright$, истинное для произвольных, но разных систем $S_1, S_2, S_3 \in \mathbb{S}$, обладающее такими свойствами:

Полнота: $(S_1 \triangleright S_2) \vee (S_2 \triangleright S_1)$

Транзитивность: $(S_1 \triangleright S_2) \wedge (S_2 \triangleright S_3) \Rightarrow (S_1 \triangleright S_3)$

Например, предположим, что операция $\triangleright$ относится к сложности, тогда свойство полноты позволяет утверждать, что всегда либо S_1 сложнее S_2, либо S_2 сложнее S_1. Точно также, если S_1 сложнее S_2 и S_2 сложнее S_3, тогда с учетом свойства транзитивности S_1 по определению сложнее S_3. Свойства полноты и транзитивности также должны соблюдаться, когда аргументами введенной операции являются не системы, а методы. Эти свойства также требуют, чтобы сложность или размер систем или методов не были одинаковыми.

Ранее введенная нами *эмпирическая система отношений* E из (1) транслируется в формальную систему отношений F [7], что и показано в следующем формальном определении:

$$F \equiv (M, IC, FR, FO) \tag{2}$$

Здесь первый аргумент $M = \{m_1, m_2, m_3, \ldots\}$ – это набор значений метрик, рассчитанных с помощью методов в V_i где для каждого метода выполняется $v_i \in V$. Формула этой метрики такова:

$$m_i = \sum_{\forall w \in V_i} \mu(w) \tag{3}$$

Множество V_i представляет собой множество методов в транзитивном замыкании t_i, построенном над m_i, а μ – это базовая метрика, например, количество строк кода или численное значение цикломатической сложности. Каждая метрика m_i является значением метрики для соответствующего метода v_i, поэтому $|M| = |V|$, то есть оба множества имеют одинаковую мощность.

Второе множество из (2) $IC = \{ic_1, ic_2, ic_3, \ldots\}$ содержит множества всех косвенно связанных методов для каждого $v_i \in V$, включая метод v_i. Каждое множество $ic_i \in IC$ совпадает с множеством V_i соответствующего транзитивного замыкания (имея в виду, что $t_i = (V_i, A_i^*)$). Множество ic_i совпадает с множеством V_i^f *прямого транзитивного замыкания* t_i^f, в этом случае используется обозначение ic_i^f, или множеству V_i^r *обратного транзитивного замыкания* t_i^r, при используемом обозначении ic_i^r.

Третий элемент формулы (2) $FR = \{fr_1, fr_2, fr_3, \ldots\}$ это конечное множество правильных *формальных отношений* между каждой парой метрик из M. Формальными отношениями, которые сравнивают размер или сложность двух методов, являются отношения <, =, и >. Например, если M включает значения метрики цикломатической сложности, и значение цикломатической сложности для метода v_i меньше, чем число цикломатической сложности другого метода v_j, тогда отношение $m_i < m_j$ истинно. Аналогично, если M – это размер кода, и число строк для v_i, больше, чем число строк v_j, тогда истинно отношение $m_i > m_j$.

В нашей работе мы рассчитываем метрики косвенных связей, детализируя их на уровне метода, и элементом, используемым в этом исчислении, является связность объектов, то есть прямое или обратное транзитивное замыкание. Наши метрики измеряют размер и сложность скрытых или многоуровневых функций. Эта скрытая сложность является результатом соблюдения таких принципов проектирования программного обеспечения, как инкапсуляция и повторное использование кода.

Например, размер или сложность любого конкретного метода можно измерить путем извлечения показателей из статической структуры его исходного кода (например, таких как параметры шаблонов FAN-IN и FAN-OUT, количество методов NOM, количество строк кода LOC и значение цикломатической сложности Маккейба CYC [36]). Набор вычисляемых метрик можно расширить, определив пользовательские функции вычисления метрик и подключив их к системе.

Следовательно, для любого метода $v_i \in V$, мы могли бы использовать конечный набор метрик, связанных с размером или сложностью метода. Чтобы представить конкретное значение индивидуальной метрики для метода v_i, мы используем выражения FAN-IN_i, FAN-OUT_i, NOM_i, LOC_i, и CYC_i. Эти отдельные показатели комбинируются для получения показателей косвенной связи. Стоит напомнить, что метод v_i косвенно связан с другим методом v_j, если существует хотя бы один путь $p = \langle v_1 \cdots v_l \rangle$, который содержит l таких методов, как $v_1 = v_i$, $v_l = v_j$, $v_k \rightarrow v_{k+1}$ для $k \in \{1 \cdots l - 1\}$, и $\{(v_1, v_2), (v_2, v_3), \cdots, (v_{l-1}, v_l)\} \subseteq t_i^f$, то есть v_j является одним из методов прямого транзитивного замыкания для v_i).

Наконец, $FO = \{fo_1, fo_2, fo_3, \ldots\}$ это конечное множество допустимых формальных бинарных операций над любой парой значений метрики из M. Формальными бинарными операциями, разрешенными для двух значений метрики $m_i, m_j \in M$ являются операции + и -. Предположим, мы выполняем операцию + над значениями метрики $m_i, m_j \in M$, которая является следствием добавления к A вызова метода (v_i, v_j). Здесь мы предполагаем, что цикл при этом не возникаеи. С одной стороны, если $ic_i \cap ic_j = \emptyset$ (при $ic_i = V_i$ и $t_i = (V_i, A_i^*)$), то есть добавленные значения метрики принадлежат несвязанным методам $v_i, v_j \in V$, то новое значение метрики для v_i равно $m_i' = m_i + m_j$. С другой стороны, если $ic_i \cap ic_j = ic_j$, то есть все методы из ic_j входят ic_i, новые методы не добавляются к ic_i и $m_i' = m_i$. Следовательно, значение m_i' находится в диапазоне $[m_i \ldots m_i + m_j]$, а значение $ic_i \cap ic_j \neq \emptyset \wedge ic_i \cap ic_i \neq ic_j$ истинно, то есть только подмножество ic_j

находится в ic_i. Аналогичные рассуждения приводят к выводу, что $m_i' \in [m_j \dots m_i + m_j]$, когда к A вместо ребра (v_i, v_j) добавляется ребро (v_j, v_i).

Эмпирическая система E преобразуется в формальную систему F путем реализации метрики μ. Каждому элементу v_i из E предназначен соответствующий элемент m_i в F, то есть, $m_i = \mu_i = \mu(v_i)$. Выбранный метод трансляции гарантирует, что понятия, лежащие в основе эмпирических отношений, остаются неизменными. Подобное преобразование позволяет автоматизировать поиск и сравнение самых сложных или крупных методов, которые, вероятно, имеют больше уровней зависимостей. Кроме того, становится возможным автоматически находить те методы, изменение которых рискованно из-за высокого уровня их повторного использования.

4. Расчетные критерии метрик

Вейкер в работе [34] предлагает систему оценок, обладающую девятью свойствами. Предложенный нами подход призван создать основу для измерения и сравнения одних предложенных метрик с другими. Таким образом, данное исследование представляет собой адаптацию свойств системы оценки Вейкер. Полученные данные перечислены ниже.

Свойство 1. Для любого метода $v_i \in V$ и метрики μ, должен существовать другой метод $v_j \in V$ такой, что $\mu(v_i) \neq \mu(v_j)$. Следовательно, значение метрики не будет одинаковым для всех методов системы. Метрика, которая считает все методы одинаковыми по размеру или сложности, бесполезна.

Свойство 2. Требуется, чтобы существовало только конечное число методов, имеющих одно и то же значение метрики. Поскольку каждая программная система всегда имеет лишь конечное число методов, мы решили проблему с циклами в графе G, превратив сильно связные компоненты (strongly connected components, SCC) в одиночные метаузлы SCC. Значения метрик размера и сложности SCC становятся суммой значений метрик их компонентов. Следовательно, каждый метод будет иметь только одно значение для каждой метрики, и этому свойству будет удовлетворять любой измеряемый метод.

Свойство 3. Два разных метода $v_i, v_j \in V$ могут иметь $\mu(v_i) = \mu(v_j)$. Другими словами, два разных метода из одной и той же системы могут иметь одно и то же значение метрики (то есть оба метода имеют одинаковый размер или сложность, в зависимости от метрики μ).

Свойство 4. В случае, если два разных метода $v_i, v_j \in V$ реализуют одну и ту же функциональность, то не обязательно истинно, что $\mu(v_i) = \mu(v_j)$. Значения метрики зависят от деталей реализации соответствующих методов. Для расчета значений метрик не имеет значения то факт, что оба метода генерируют один и тот же набор выходных данных и при одном и том же наборе входных данных имеют в системе одинаковые побочные эффекты.

Свойство 5. Для всех различных методов $v_i, v_j \in V$ должно выполняться: $\mu(v_i) \leq \mu(v_i \oplus v_j) \wedge \mu(v_j) \leq \mu(v_i \oplus v_j)$. Здесь, $v_i \oplus v_j$ представляет собой перестроение графов косвенной связи v_i и v_j (то есть ICG_i и ICG_j), которое получается добавлением связи для двух методов, в каждом из двух подграфов. Это означает, что ложны оба отношения $m_i > m_i'$ и $m_j > m_j'$, то есть новое значение метрики для объединения двух подграфов косвенных связей всегда должно быть больше или равно метрике любого из подграфов.

Свойство 6. Пусть $v_i, v_j, v_k \in V$ – три разных метода в системе, тогда $\mu(v_i) = \mu(v_j) \nRightarrow \mu(v_i \oplus v_k) = \mu(v_j \oplus v_k)$. Может случиться, что значение метрики объединения графов ICG_i и ICG_k будет отличаться от значения метрики графа, полученного путем объединения ICG_j и ICG_k.

Свойство 7. Размер или сложность метода v_i должны влиять на расположение или порядок путей вызова ICG_i. Пусть $(v_i, v_j), (v_j, v_k) \in A_i$ – два разных вызова метода в ICG_i, и предположим, что другого пути от v_i к v_j не существует. Новый граф ICG_i' для v_i' получается в результате изменения порядка этих вызовов на $(v_j', v_i'), (v_i', v_k)$. Тогда должно быть

истинно, что $\mu(v_i) \neq \mu(v_i')$, учитывая, что теперь $v_j' \notin V_i'$. Это свойство обретает смысл в контексте метрик, предназначенных для измерения сложности данных или потока управления.

Свойство 8. Если метод v_i' это переименованный метод $v_i \in V$, то $\mu(v_i) = \mu(v_i')$. В данном случае переименование – это последовательная систематическая замена всех вхождений идентификатора в одной области определения на другой, ранее не использовавшийся идентификатор. Это свойство выполняется всякий раз, когда мы применяем замену к подграфу косвенных связей любого метода.

Свойство 9. Могут существовать разные методы $v_i, v_j \in V$, такие, что $\mu(v_i) + \mu(v_j) < \mu(v_i \oplus v_j)$. В некоторых случаях объединенный граф может быть больше или сложнее, чем сумма их размеров или сложностей. Это свойство пытается отразить тот факт, что между комбинированными зависимостями может возникать некоторое взаимодействие.

Кроме того, мы высказываем некоторые предположения, которые необходимы для аналитической оценки предложенных метрик. Первое предположение касается статистического распределения значений метрик, а остальные два – мощности объединенных методов посредством реструктуризации.

Предположение 1. Пусть $v_i \in V$ – произвольный метод из системы S, и

$FAN\text{-}IN_i$ = Количество методов, вызывающих v_i,
$FAN\text{-}OUT_i$ = Количество методов, вызываемых из v_i,
LOC_i = Количество строк кода в v_i,
CYC_i = Цикломатическая сложность v_i.

Значения $FAN\text{-}IN_i$, $FAN\text{-}OUT_i$, LOC_i, и CYC_i, являются дискретными случайными переменными. Каждая из этих переменных имеет свою функцию распределения. Для каждого $v_i \in V$, все значения $FAN\text{-}IN_i$ независимы и одинаково распределены (independent and identically distributed, i.i.d.), это же относится ко всем значениям $FAN\text{-}OUT_i$, LOC_i, и CYC_i. В этом плане количество вызовов методов, число строк кода и цикломатическая сложность подчиняются неочевидному статистическому распределению. Более того, невозможно предсказать количество вызовов методов, число строк кода и цикломатическую сложность одного метода, основываясь на знании количества вызовов методов, числа строк кода и цикломатической сложности другого метода в той же системе.

Предположение 2. Любые два метода v_i и v_j могут иметь конечное число общих методов в ICG_i^f и ICG_j^f, в том плане, что объединение обоих методов в один приведет к тому, что какой-нибудь из общих методов может перестать быть общим (то есть сумма мощностей обоих подграфов будет больше или равна мощности объединенных подграфов хрупкости)

$$\left|ICG_i^f \cup ICG_j^f\right| = \left|ICG_i^f\right| + \left|ICG_j^f\right| - \left|ICG_i^f \cap ICG_j^f\right|.$$

Предположение 3. Аналогично, любые два метода v_i и v_j могут иметь конечное число общих методов в ICG_i^r и ICG_j^r, в том плане, что объединение обоих методов в один приведет к тому, что какой-нибудь из общих методов может перестать быть общим (то есть сумма мощностей обоих подграфов будет больше или равна мощности объединенных подграфов жесткости)

$$\left|ICG_i^r \cup ICG_j^r\right| = |ICG_i^r| + \left|ICG_j^r\right| - \left|ICG_i^r \cap ICG_j^r\right|.$$

5. Сбор эмпирических данных

В этом разделе объясняется метод, который реализован в нашем программном инструменте для извлечения данных из исходного кода программных систем и расчета количественных характеристик косвенных связей. В разделе также представлен и описывается набор данных программных систем, используемых при проверке метрик.

5.1 Алгоритм

Разработанный нами инструмент строит графы косвенных связей (ICG) для каждой функции, обнаруженной в программной системе. Программа написана на языке C, использует платформу с открытым исходным кодом и для компиляции исходного кода программ и построения его *абстрактного синтаксического дерева* (Abstact Syntax Tree, AST) с семантическими привязками, выполнения статического анализа AST и создания графа зависимости программы (Program Dependence Graph, PDG) использует технологии Roslyn и .NET Compiler Platform (см. алгоритм 1 и рис. 1). Узлы PDG – это методы из AST, а его ребра – это полиморфные вызовы между этими методами. Полиморфный вызов – это любой прямой вызов между двумя методами, любой вызов, который может быть выполнен через реализацию интерфейса, или любой вызов, уточненный схемой наследования.

PDG может иметь циклы, то есть циклические пути вызова между двумя или более методами. Другое название подграфов, образованных этими циклами, – сильно связанные компоненты (Strongly Connected Components, SCC). Методы, определенные вне цикла, могут вызывать один или несколько методов цикла. Аналогично, методы в цикле могут вызывать методы вне цикла. Тем самым, трудно гарантировать, что циклы будут иметь не более одной точки входа и одной точки выхода. Наш инструмент реализует алгоритм Габоу [37] для преобразования PDG в ориентированный ациклический граф путем выявления сильно связанных компонентов и замены каждого из них одним свернутым метаузлом. Весовое значение этих свернутых компонентов SCC представляет собой сумму весовых значений его методов. Каждый метод имеет несколько различных весовых характеристик: FAN-IN, FAN-OUT, NOM, LOC и CYC.

```
1: Input C – множество файлов исходного кода
2: Output M – множество извлеченных метрик
3: Output IC – множества косвенно связанных методов
4:
5: DC ← ∅                          ▷ DC: Граф анализа вызовов методов, иерархии классов и реализации интерфейсов
6:
7: for all S ∈ C do                                                          ▷ S: Файл исходного кода
8:         ASTs ← Generate AST from S                          ▷ AST_S: Абстрактное синтаксическое дерево из S
9:         for all references (i, j) ∈ AST_S do
10:            Add Reference (i, j) to Graph DC    ▷ Преобразование вызовов полиморфных методов в фактические
11:        end for
12: end for
13:
14: SCC ← Запуск алгоритма Габоу (поиск SCC в DC)          ▷ SCC: множество сильно связанных компонентов
15: DAG ← Свертка SCC на основе DC и SCC                        ▷ DAG: Направленный ациклический граф
16: IC ← Поиск в ширину на множестве методов IC из DAG       ▷ IC: Множество косвенно связанных методов
17: M ← Извлечение метрик с использованием DAG и IC                    ▷ M: Множество метрик
18:
19: return ⟨ M, IC⟩
```

Алгоритм 1. Алгоритм извлечения компонентов M и IC для формальной системы отношений F
Algorithm 1. Algorithm used by our tool to extract components M and IC for the formal relational system F

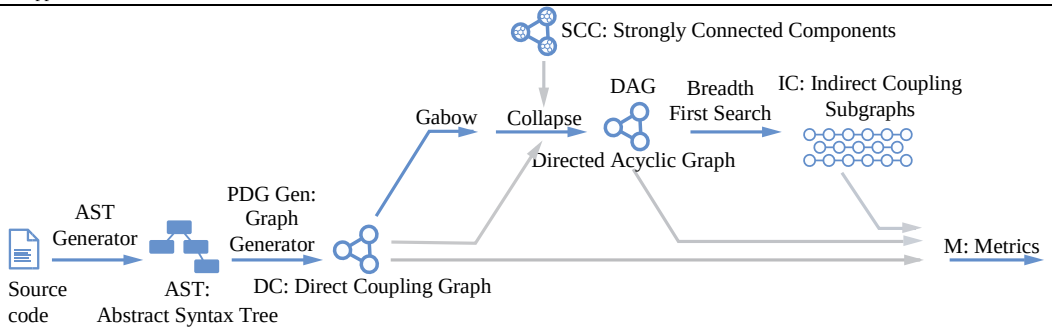

Рис. 1. Алгоритм извлечения метрик и подграфов косвенных связей из исходного кода.
Fig. 1. Algorithm used to extract metrics and indirect coupling subgraphs from source code.

Наш инструментарий рассматривает ациклический граф DAG со свернутыми SCC и применяет поиск в ширину, составляя для каждого метода в системе множество транзитивных замыканий *T* и множество методов с косвенными связями *IC*.

Набор транзитивных замыканий *T* содержит прямые транзитивные замыкания $\boldsymbol{t_i^f}$ для каждого метода $\boldsymbol{v_i} \in \boldsymbol{V}$ для расчета хрупкости или обратные транзитивные замыкания $\boldsymbol{t_i^r}$ для расчета жесткости. Множество $\boldsymbol{IC}$ методов с косвенными связями содержит множества косвенных связей $\boldsymbol{ic_i}$ для каждого $\boldsymbol{v_i}$. Заметьте, что $\boldsymbol{ic_i}$ равно $\boldsymbol{ic_i^f}$ в метриках хрупкости и равно $\boldsymbol{ic_i^r}$ в метриках жесткости, и что $\boldsymbol{ic_i^f} = \boldsymbol{V_i^f}$ – это множество узлов $\boldsymbol{t_i^f} = \left(\boldsymbol{V_i^f}, \boldsymbol{A_i^{f*}}\right)$, а $\boldsymbol{ic_i^r} = \boldsymbol{V_i^r}$ – это множество узлов $\boldsymbol{t_i^r} = (\boldsymbol{V_i^r}, \boldsymbol{A_i^{r*}})$. Наш инструмент вычисляет метрики $\boldsymbol{m_i} \in \boldsymbol{M}$, используя весовые значения элементов в $\boldsymbol{ic_i}$.

5.2 Набор данных

Программы, которые использовались для проверки вычисленных метрик, представляют собой 11 систем, написанных на языке C#. Шесть из них относятся к категории промышленных, а остальные пять – это программы с открытым исходным кодом. В табл. 1 для этих систем приведены сведения о количестве методов или узлов в графе *G* (NOM), общее количество строк кода (LOC), общая комбинированная цикломатическая сложность (CYC), число классов (Classes) и вызовов методов или ребер в графе *G* (Calls). В этой таблице также указано общее количество косвенных связей в *G* (Paths), максимальная и средняя длина таких связей, а также среднее количество строк кода всех методов в каждой системе. Системы отсортированы в порядке убывания общего количества косвенных связей в каждой системе (Paths). Значения в столбцах NOM, LOC, CYC, Classes и Calls – это наши подсчеты, они не учитывают косвенную связь между методами.

Промышленные системы созданы двумя компаниями-разработчиками программного обеспечения, названными здесь Компания 1 и Компания 2, чтобы соблюсти соглашения о неразглашении информации. Исследуются системы Система 1, Система 4, Система 5 и Система 6, разработанные Компанией 1, а также Система 2 и Система 3 Компании 2. Размер этих программ колеблется от 4606 до 188463 строк текста (LOC).

Нами также анализировались приложения с открытым исходным кодом, к которым относятся MongoDB C# Driver 2.9 [38], Json.NET 12.0.1 [39], .NET Micro Framework 4.4 [40], NodeJS Tools [41] и Neo4j .NET Driver 2.0 [42], размер которых находится в диапазоне от 3403 до 67633 строк текста (LOC). По мере увеличения количества строк кода в системах также увеличивается количество классов и количество методов, за исключением приложения O_2, у которого меньше классов, чем у приложения O_5, но в среднем более крупные методы.

Табл. 1. Численные характеристики промышленных систем и систем с открытым исходным кодом
Table 1. Size properties of industrial and open source systems

ID	Системное имя	NOM	LOC	CYC	Classes	Calls	Paths	Длина (max)	Длина (средняя)	$\frac{LOC}{Methods}$
I_1	System 1	2,832	17,020	6,062	842	3,290	73,667	15	7	6.01
I_2	System 2	20,362	188,463	55,693	2,633	23,351	51,202	14	2	9.26
I_3	System 3	18,525	177,050	42,598	3,507	23,893	42,030	9	3	9.56
I_4	System 4	3,786	29,077	10,293	1,563	347	3,760	8	1	7.68
I_5	System 5	1,065	6,062	2,102	336	349	1,037	8	1	5.69
I_6	System 6	825	4,603	1,542	251	386	739	8	1	5.58
O_1	MongoDB C# Driver	8,034	37,181	16,128	1,415	16,073	20,751,439	35	14	4.63
O_2	Json.NET	1,574	10,185	5,401	167	3,104	11,013,586	29	16	6.47
O_3	.NET Micro Framework	9,038	67,633	25,217	1,824	10,823	57,428	22	8	7.48
O_4	NodeJS Tools	3,288	18,507	8,731	596	3,185	22,893	18	8	5.63
O_5	Neo4j .NET Driver	1,018	3,403	1,523	210	1,150	8,324	22	10	3.34

6. Расчетные критерии метрик

В этом разделе описание каждой предлагаемой метрики выделено в отдельный подраздел. На вопрос (RQ1), поставленный в начале статьи, отвечают подразделы “Определение”, “Теоретические основы” и “Аналитические оценки”. На вопрос (RQ2) отвечают подразделы с заголовками: “Важные соображения” и “Экспериментальные данные”.

6.1 Вычисление показателя хрупкости методов (FNOM)

Метрика FNOM для данного метода показывает количество методов в подграфе хрупкости.

6.1.1 Определение

Пусть $\boldsymbol{v_i} \in \boldsymbol{V}$ – это метод, $\boldsymbol{\mu} = \text{FNOM}$, и $\boldsymbol{ic_i} = \{\boldsymbol{v_1} \cdots \boldsymbol{v_n}\}$ – множество всех методов в $\boldsymbol{ic_i^f}$ (то есть все методы в графе хрупкости косвенных связей метода $\boldsymbol{v_i}$, включая сам этот метод). Тогда:

$$m_i = \mu_i = \text{FNOM}_i = \sum_{k=1}^{n} \text{NOM}_k \tag{4}$$

6.1.2 Теоретические основы

В отношении конкретного метода метрика FNOM соответствует расширению метрики сложности FAN-OUT. FNOM – это FAN-OUT, рассчитываемая для подграфа хрупкости косвенных связей этого метода (таким образом, получаемое значение соответствует количеству методов, косвенно вызываемых из данного, включая сам этот метод). FNOM включает в себя значения метрики FAN-OUT всех методов в подграфе хрупкости метода. Эта метрика учитывает скрытые зависимости на этапе, возникшие при проектировании, и применяется при управлении размером и сложностью, а также для следования принципам инкапсуляции.

6.1.3 Аналитические оценки

Пусть $\boldsymbol{v_i}, \boldsymbol{v_j} \in \boldsymbol{V}$ два разных произвольных метода из системы $\boldsymbol{E}$. Предполагается, что переменные FAN-OUT_i и FAN-OUT_j являются i.i.d. (см. Предположение 1). Аналогичные

предположения делаются для переменных $\boldsymbol{m_i} = \text{FNOM}_i$ и $\boldsymbol{m_j} = \text{FNOM}_j$. Соответственно с ненулевой вероятностью $\boldsymbol{\mu_i \neq \mu_j}$, тем самым удовлетворяется Свойство 1. Свойство 2 также выполняется (см. свойства в разделе 4). Существует ненулевая вероятность того, что $\exists\, \boldsymbol{v_k} \in \boldsymbol{V} \mid \boldsymbol{\mu_i = \mu_k}$, следовательно, Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет количество вызовов методов в его подграфе хрупкости, поскольку это число в большей степени зависит от проектных решений, не зависящих от этой функциональности, поэтому Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i = m_i}$ и $\boldsymbol{\mu_j = m_j}$, тогда значение метрики объединения обоих методов равно $\boldsymbol{\mu(v_i \oplus v_j) = m_i + m_j - \delta}$, где $\boldsymbol{\delta}$ количество общих методов в графах $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Следовательно, $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_i}$ и $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_j}$, что гарантирует выполнение Свойства 5. Далее, пусть $\boldsymbol{\mu_i = \mu_j}$, и $\exists\, \boldsymbol{v_k} \in \boldsymbol{V}$ такой, что $\boldsymbol{ic_k^f}$ имеет $\boldsymbol{\beta}$ методов, общих с $\boldsymbol{ic_i^f}$ (см. Предположение 2) и $\boldsymbol{\delta}$ методов, общих с $\boldsymbol{ic_j^f}$, где $\boldsymbol{\beta \neq \delta}$. Тогда

$$\boldsymbol{\mu(v_i \oplus v_k) = \mu_i + \mu_k - \beta}$$
$$\boldsymbol{\mu(v_j \oplus v_k) = \mu_j + \mu_k - \delta}$$

Следовательно, $\boldsymbol{\mu(v_i \oplus v_k) \neq \mu(v_j \oplus v_k)}$, то есть Свойство 6 выполняется. Свойства 7 и 8 выполняются (см. свойства в разделе 4). Пусть для любых двух методов $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$ число $\boldsymbol{\gamma}$ – это количество общих методов в графах $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$, а $\boldsymbol{\delta}$ – это количество новых методов, возникших в результате реструктуризации, необходимой для объединения графов $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$ и получения графа $\boldsymbol{ic_{i'}^f}$. Тогда если $\boldsymbol{\delta > \gamma}$, то истинно отношения $\boldsymbol{m_i + m_j - \gamma + \delta > m_i + m_j}$ (то есть $\boldsymbol{\mu(v_i \oplus v_j) > \mu(v_i) + \mu(v_j)}$ при заданных $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$). Следовательно, Свойство 9 выполняется.

6.1.4 Важные соображения

Ниже приводятся некоторые соображения, которые полезно учитывать в отношении FNOM:

- FNOM – это более полная метрика сложности метода, чем FAN-OUT. FNOM – своего рода рекурсивный вариант метрики FAN-OUT.
- Количество методов, решающих сформулированные в программе задачи, можно использовать для предсказания того, сколько времени и усилий потребуется для разработки, понимания (включая прослеживание методов) и сопровождения функциональности, выполняемой методом (entry point method) [7].
- Чем больше значение метрики FNOM метода, тем больше вероятность того, что любое изменение повлияет на этот метод через его зависимости (то есть, что возникнет волновой эффект ошибок, изменений, тестирования, реструктуризации).
- Методы с многочисленными зависимостями, считаются менее пригодными для повторного использования, они более специфичны для приложения.

У корневого метода метрика FAN-IN равна нулю; у листового метода равна нулю метрика FAN-OUT. Средние значения метрик для всех корневых и конечных методов всех систем приведены в табл. 2. Средние значения метрик хрупкости рассчитаны только для корневых методов, а средние значения метрик жесткости представлены только для листовых методов.

6.1.5 Экспериментальные данные

Сводная статистика по метрике FNOM во всех системах представлена в табл. 3. Помимо минимума, 1-го квартиля, медианы, 3-го квартиля и максимума для FNOM, в ней показано наиболее частое расстояние (MFD) между значениями метрик FAN-OUT и FNOM (то есть для каждого метода v_i в системе расстояние равно $D_i = \text{FNOM}_i - \text{FAN-OUT}_i$ так что MFD

будет наиболее частым D_i в системе), а также процентная доля методов со значением D_i = MFD от их общего числа. В таблице также представлен коэффициент корреляции Кендалла между значениями метрик FAN-OUT и FNOM, а также его *p-значение*.

Наиболее широко используемые статистические коэффициенты корреляции – Пирсона, Спирмена и Кендалла. Мы не могли использовать корреляцию Пирсона, поскольку она требует, чтобы данные соответствовали нормальному распределению. Мы также не могли использовать корреляцию Спирсмена, потому что она неверно ведет учет зависимостей. Мы выбрали τ-статистику Кендалла [43], используемую для оценки ранговой меры связи между двумя метриками.

Табл. 2. Средние значения метрик из методов root/leaf†*

Место для уравнения.*Table 2. Average metric values from root*/leaf† methods*

Системное имя	FNOM	FLOC	FCYC	RNOM	RLOC	RCYC
I_1	16	122	50	14	110	31
I_2	3	29	10	3	83	19
I_3	6	66	27	6	182	28
I_4	1	8	3	1	9	3
I_5	1	8	3	2	10	3
I_6	2	10	3	2	13	4
O_1	74	544	224	76	522	147
O_2	49	335	166	56	588	324
O_3	8	53	20	9	131	49
O_4	5	29	13	6	67	31
O_5	11	40	15	10	46	21

**Корневые методы для FNOM, FLOC и FCYC (Root methods for FNOM, FLOC and FCYC)*
†Листовые методы для RNOM, RLOC и RCYC (Leaf methods for RNOM, RLOC and RCYC)

Табл. 3. Сводная статистика для метрики FNOM
Table 3. Summary Statistics for the FNOM metric

Sys	MFD*	%MDF	$\tau^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
I_1	1	64	0.82	0	1	1	2	12	119
I_2	1	85	0.95	0	1	1	1	3	229
I_3	1	63	0.88	0	1	1	2	4	100
I_4	1	98	1.00	0	1	1	1	1	47
I_5	1	95	0.99	0	1	1	1	1	47
I_6	1	92	0.98	0	1	1	1	2	47
O_1	1	58	0.80	0	1	1	2	31	560
O_2	1	49	0.73	0	1	1	3	23	689
O_3	1	65	0.85	0	1	1	2	6	300
O_4	1	75	0.90	0	1	1	1	3	234
O_5	1	70	0.88	0	1	1	1	4	153

**MFD: наиболее частое расстояние между FNOM и FAN-OUT (MFD: most frequent distance between FNOM and FAN-OUT)*

†τ: коэффициент корреляции Кендалла между FNOM и FAN-OUT (τ: Kendall correlation coefficient between FNOM and FAN-OUT)

Функции плотности [44] метрики FNOM для всех систем представлены на рис. 2, а линейные графики со значениями метрики для FNOM и FAN-OUT для всех методов [45] в каждой системе, ранжированной по FAN-OUT, показаны на рис. 3 (каждое значение по ординате линейного графика представляет значения метрик FNOM и FAN-OUT, а значения по оси абсцисс представляют ранг методов, упорядоченных по FAN-OUT, а затем по FNOM, те значения, для которых разница между FNOM и FAN-OUT составляет ровно MFD, не отображаются).

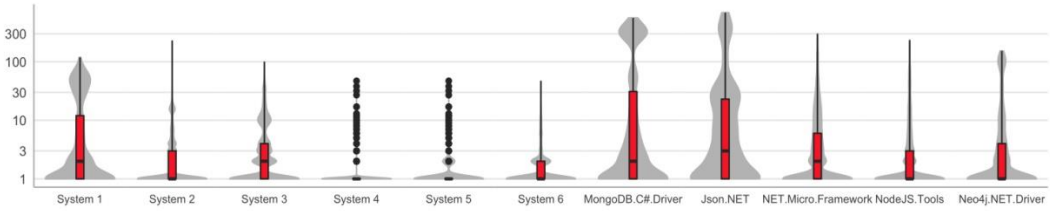

Рис. 2. Функция плотности для FNOM для всех систем
Рис. 2. Density function for FNOM from all systems

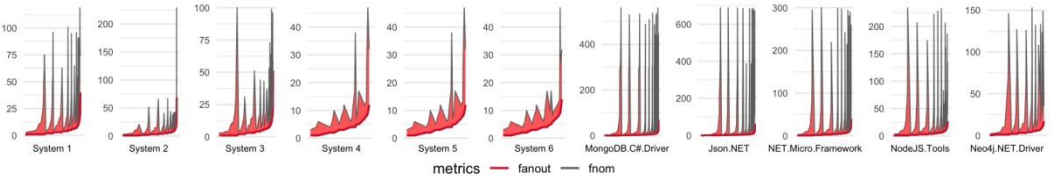

Рис. 3. Расстояние между значениями FNOM и FAN-OUT для ранжированных методов всех систем
Fig. 3. Distance between FNOM and FAN-OUT values for the ranked methods from all systems

6.2 Вычисление показателя хрупкости числа строк текста (FLOC)

FLOC — это совокупное количество строк кода всех методов в подграфе хрупкости данного метода.

6.2.1 Определение

Пусть $\boldsymbol{v_i} \in \boldsymbol{V}$ является методом, $\boldsymbol{\mu} = \text{FLOC}$, и $\boldsymbol{ic_i} = \{\boldsymbol{v_1} \cdots \boldsymbol{v_n}\}$ множество всех методов в $\boldsymbol{ic_i^f}$, тогда:

$$m_i = \mu_i = \text{FLOC}_i = \sum_{k=1}^{n} \text{LOC}_k \quad (5)$$

6.2.2 Теоретические основы

Метрика FLOC – это, по сути, расширение метрики LOC (число строк текста), вычисленное для некоторого метода. FLOC – это метрика LOC, применяемая к подграфу хрупкости косвенных связей метода (она соответствует совокупному количеству строк текста во всех методах, которые данный метод может прямо или косвенно вызывать, а также в самом этом методе). Эта метрика также демонстрирует зависимости, не проявившиеся на этапе проектирования, это помогает управлять сложностью и размером программ, а также следовать принципам инкапсуляции.

6.2.3 Аналитические оценки

Пусть $\boldsymbol{v_i, v_j} \in \boldsymbol{V}$ – два разных произвольных метода из системы $\boldsymbol{E}$. Предполагается, что переменные $\mathbf{LOC_i}$ и $\mathbf{LOC_j}$ обладают свойством i.i.d. (см. Предположение 1), то же самое предполагается для переменных $\boldsymbol{m_i} = \text{FLO}\mathbf{C_i}$ и $\boldsymbol{m_j} = \text{FLO}\mathbf{C_j}$; следовательно $\boldsymbol{\mu_i \neq \mu_j}$, с ненулевой вероятностью, и выполняется Свойство 1. Свойство 2 также выполняется (см. свойства в разделе 4). Существует ненулевая вероятность того, что $\exists\, \mathbf{v_k} \in \boldsymbol{V} \mid \boldsymbol{\mu_i = \mu_k}$, следовательно Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет общее количество строк кода во всех методах в его подграфе хрупкости, поскольку размер программы – это проектное решение, не зависящее от реализуемой функциональности, следовательно, Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i = m_i}$ и $\boldsymbol{\mu_j = m_j}$, тогда $\boldsymbol{\mu(v_i \oplus v_j) = m_i + m_j - \delta}$, где $\boldsymbol{\delta}$ – общее количество строк кода в общих методах между $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Следовательно, $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_i}$ и $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_j}$, поэтому Свойство 5 выполняется. Далее, пусть $\boldsymbol{\mu_i = \mu_j}$, и $\exists\, \boldsymbol{v_k} \in \boldsymbol{V}$ такой, что $\boldsymbol{ic_k^f}$ имеет $\boldsymbol{\beta}$ строк текста в общих с $\boldsymbol{ic_i^f}$ методах (см. Условие 2) и $\boldsymbol{\delta}$ строк текста в общих методах с $\boldsymbol{ic_j^f}$, где $\boldsymbol{\beta \neq \delta}$. Тогда

$$\boldsymbol{\mu(v_i \oplus v_k) = \mu_i + \mu_k - \beta}$$
$$\boldsymbol{\mu(v_j \oplus v_k) = \mu_j + \mu_k - \delta}$$

Следовательно, $\boldsymbol{\mu(v_i \oplus v_k) \neq \mu(v_j \oplus v_k)}$ и Свойство 6 выполняется. Свойства 7 и 8 выполняются. Для любых двух методов $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$, пусть $\boldsymbol{\gamma}$ будет значением совокупного числа строк общих методов в графах $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$, и пусть $\boldsymbol{\delta}$ будет совокупным значением числа строк новых методов, добавленных в результате реструктуризации при объединении графов $\boldsymbol{ic_i^f}$ и $\boldsymbol{ic_j^f}$ в граф $\boldsymbol{ic_{i'}^f}$. Если $\boldsymbol{\delta > \gamma}$, то $\boldsymbol{m_i + m_j - \gamma + \delta > m_i + m_j}$ (то есть $\boldsymbol{\mu(v_i \oplus v_j) > \mu(v_i) + \mu(v_j)}$ при заданных $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$). Следовательно, Свойство 9 выполняется.

6.2.4 Важные соображения

Соображения, рассматриваемые в отношении FLOC, перечислены ниже.

- FLOC — это более точный показатель размера метода, чем LOC. FLOC образуется суммированием нужных значений метрики LOC.
- Совокупный размер всех методов, выполняющих некоторую функциональность, также может помочь спрогнозировать необходимое время и усилия для реализации, понимания (включая поиск зависимостей) и изменения требований, выполняемых методом, являющимся начальной точкой графа [7].
- Более высокие значения метрики FLOC метода увеличивают вероятность того, что через любую из его зависимостей (то есть каскадные ошибки или изменения, тестирование, реструктуризация) метод может быть изменен. Например, вероятность того, что какое-либо изменение повлияет на монолитную программу, равна 1.
- Методы, которые имеют больше зависимостей, предположительно, сложнее использовать повторно, они более специфичны для конкретного приложения.

6.2.5 Экспериментальные данные

Сводная статистика по метрике FLOC для всех систем представлена в табл. 4. В этой таблице показаны минимум, 1-й квартиль, медиана, 3-й квартиль и максимум для метрики FLOC. Она также показывает наиболее частое расстояние MFD между значениями метрик LOC и FLOC и долю методов со значением метрики, равным MFD, в общем количестве методов. В ней

также представлен коэффициент корреляции Кендалла между значениями метрик LOC и FLOC, а также его *p-значение*.

Средние значения метрик FLOC для всех корневых методов всех систем представлены в табл. 2.

Функции плотности метрики FLOC для всех систем представлены на рис. 4, а линейные диаграммы со значениями метрики FLOC и LOC для методов каждой системы, ранжированной по LOC, представлены на рис. 5.

Табл. 3. Сводная статистика для метрики FLOC
Table 3. Summary Statistics for the FLOC metric

Sys	MFD*	%MDF	$\tau^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
I_1	0	47	0.33	0	1	3	6	96	990
I_2	0	65	0.85	0	1	1	4	15	20,964
I_3	0	45	0.63	0	1	1	9	37	4,860
I_4	0	94	0.93	0	1	1	6	11	291
I_5	0	80	0.71	0	1	3	4	8	291
I_6	0	71	0.56	0	1	3	4	7	300
O_1	0	40	0.48	0	1	1	8	164	4,205
O_2	0	34	0.35	0	1	3	16	92	5,621
O_3	0	49	0.65	0	1	1	5	32	2,940
O_4	0	60	0.70	0	1	1	4	17	1,589
O_5	0	56	0.59	0	1	1	3	17	555

**MFD: наиболее частое расстояние между FLOC и LOC (MFD: most frequent distance between FLOC and LOC)*

$^{\dagger}\tau$: коэффициент корреляции Кендалла между FLOC и LOC (τ: Kendall correlation coefficient between FLOC and LOC)

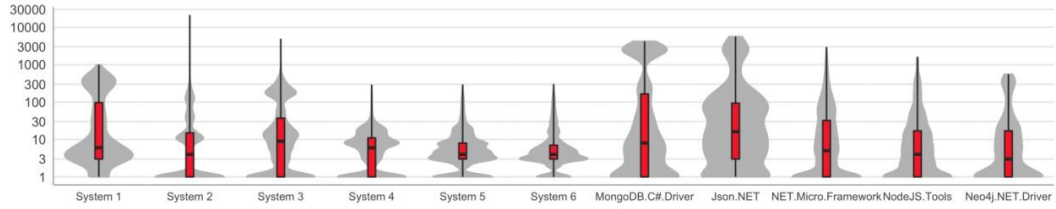

Рис. 4. Функция плотности для FLOC для всех систем
Fig. 4. Density function for FLOC from all systems

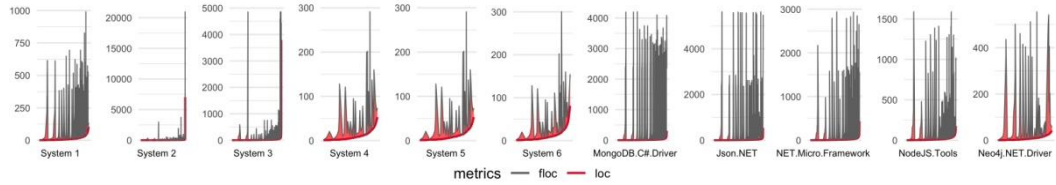

Рис. 5. Расстояние между значениями FLOC и LOC для ранжированных методов всех систем
Fig. 5. Distance between FLOC and LOC values for the ranked methods from all systems

6.3 Вычисление показателя хрупкости цикломатической сложности (FCYC)

Метрика FCYC – это число, показывающее значение консолидированной цикломатической сложности всех методов подграфа хрупкости данного метода.

6.3.1 Определение

Пусть $v_i \in V$ – метод, μ = FCYC, и $ic_i = \{v_1 \cdots v_n\}$ множество всех методов в ic_i^f, тогда:

$$m_i = \mu_i = \text{FCYC}_i = \sum_{k=1}^{n} \text{CYC}_k - (n-1) \quad (6)$$

Компонент $(n-1)$ в формуле корректирует результирующее число цикломатической сложности. Результат этой формулы представляет собой подстановку всех методов (их расширение подобно макросам) в корневой метод.

6.3.2 Теоретические основы

FCYC соответствует расширению метрики сложности CYC для одиночного метода. Метрика FCYC – это та же метрика CYC, но в применении к подграфу хрупкости косвенных связей этого метода. Оно соответствует сумме цикломатических сложностей всех методов, которые косвенно вызывает данный метод, включая его самого. Когда текст вызываемого метода встраивается в текст вызывающего метода, CYC результирующего кода равен сумме соответствующих им значений CYC минус 1. В результате вычитаемое в формуле (6) компенсирует лишние $(n-1)$ единиц. Эта метрика также учитывает зависимости, скрытые на этапе проектирования, что позволяет управлять размером и сложностью, а также следовать правилам инкапсуляции.

6.3.3 Аналитические оценки

Пусть $\boldsymbol{v_i, v_j \in V}$ – два разных произвольных метода системы $\boldsymbol{E}$. Переменные $\mathbf{CYC_i}$ и $\mathbf{CYC_j}$ будут i.i.d. (см. Предположение 1), то же самое мы предполагаем относительно переменных $\boldsymbol{m_i} =$ FCY$\mathbf{C_i}$ и $\boldsymbol{m_j} =$ FCY$\mathbf{C_j}$; следовательно, $\boldsymbol{\mu_i \neq \mu_j}$ с ненулевой вероятностью. Следовательно, Свойство 1 выполняется. Свойство 2 также выполняется (см. свойства в разделе 4). Существует ненулевая вероятность, что $\boldsymbol{\exists\, v_k \in V \mid \mu_i = \mu_k}$, соответственно, Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет общее совокупное число цикломатической сложности для всех методов в его подграфе хрупкости, поскольку это проектное решение, не зависящее от функциональности, следовательно, Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i = m_i}$ и $\boldsymbol{\mu_j = m_j}$, тогда $\boldsymbol{\mu(v_i \oplus v_j) = m_i + m_j - \delta}$, где $\boldsymbol{\delta}$ – общее совокупное число цикломатической сложности для общих методов в графах $\boldsymbol{if_i^f}$ и $\boldsymbol{ic_j^f}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Тогда $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_i}$ и $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_j}$, соответственно, Свойство 5 выполняется. Теперь, пусть $\boldsymbol{\mu_i = \mu_j}$, и $\boldsymbol{\exists\, v_k \in V}$ такой, что граф $\boldsymbol{ic_k^f}$ имеет совокупное цикломатическое число сложности $\boldsymbol{\beta}$ из общих методов с графом $\boldsymbol{ic_i^f}$ (см. Предположение 2) и совокупное число цикломатической сложности $\boldsymbol{\delta}$ с графом $\boldsymbol{ic_j^f}$, где $\boldsymbol{\beta \neq \delta}$. Тогда

$$\boldsymbol{\mu(v_i \oplus v_k) = \mu_i + \mu_k - \beta}$$
$$\boldsymbol{\mu(v_j \oplus v_k) = \mu_j + \mu_k - \delta}.$$

Таким образом, $\mu(v_i \oplus v_k) \neq \mu(v_j \oplus v_k)$, и Свойство 6 выполняется. Свойства 7 и 8 также выполняются (см. раздел 4). Пусть γ – совокупное значение метрики CYC методов для двух произвольных методов v_i и v_j, общих для графов ic_i^f и ic_j^f, и пусть δ будет накопленным значением метрики CYC новых программ, написанных для объединения графов ic_i^f и ic_j^f в

граф ic_i^f. Если $\delta > \gamma$, то ясно, что $m_i + m_j - \gamma + \delta > m_i + m_j$ (то есть, $\mu(v_i \oplus v_j) > \mu(v_i) + \mu(v_j)$ для заданных v_i и v_j). Следовательно, Свойство 9 выполняется.

6.3.4 Важные соображения

Соображения, рассматриваемые в отношении FCYC, перечислены ниже.

- FCYC — это более точный количественный показатель цикломатической сложности, чем CYC, поскольку он принимает во внимание значения CYC всех зависимых компонентов.
- Совокупная цикломатическая сложность всех методов, выполняющих некоторую работу, также может помочь спрогнозировать необходимое время и усилия для реализации, понимания (включая исследование подграфа хрупкости) и развития функциональности исходного метода [7].
- Методы с более высокими значениями метрики FCYC из-за своих зависимостей будут скорее всего более чувствительны к любым модификациям.
- Методы, с более сложными зависимостями, по-видимому, будет сложнее использовать повторно, они более специфичны для конкретного приложения.

6.3.5 Экспериментальные данные

Сводная статистика по показателю FCYC для всех систем представлена в табл. 5. В этой таблице показаны минимум, 1-й квартиль, медиана, 3-й квартиль и максимум для FCYC. Таблица также показывает наиболее часто отмеченное расстояние MFD между значениями метрик CYC и FCYC, долю методов, имеющих именно такое расстояние MFD, среди всех методов и коэффициент корреляции Кендалла между значениями метрик CYC и FCYC, а также *p-значение*.
Средние значения метрики FCYC для всех корневых методов всех систем показаны в табл. 2.

Табл. 5. Сводная статистика для метрики FCYC
Table 5. Summary Statistics for the FCYC metric

Sys	MFD*	%MDF	$\tau^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
I_1	0	51	0.4	0	1	2	2	34	314
I_2	0	67	0.7	0	1	1	2	8	1,706
I_3	0	55	0.62	0	1	1	3	12	592
I_4	0	95	0.94	0	1	1	2	3	89
I_5	0	82	0.8	0	1	1	2	2	89
I_6	0	75	0.73	0	1	1	2	2	106
O_1	0	50	0.38	0	1	1	3	53	1,718
O_2	0	37	0.29	0	1	2	8	50	2,940
O_3	0	61	0.63	0	1	1	2	11	1,208
O_4	0	67	0.68	0	1	1	2	8	671
O_5	0	63	0.54	0	1	1	2	6	231

**MFD: наиболее частое расстояние между FCYC и CYC (MFD: most frequent distance between FCYC and CYC)*

†τ: коэффициент корреляции Кендалла между FCYC и CYC (τ: Kendall correlation coefficient between FCYC and CYC)

Функции плотности метрики FCYC для всех систем показаны на **р**ис**.** 6, а линейные диаграммы со значениями метрик FCYC и CYC для методов каждой системы, ранжированной по CYC, представлены на рис. 7.

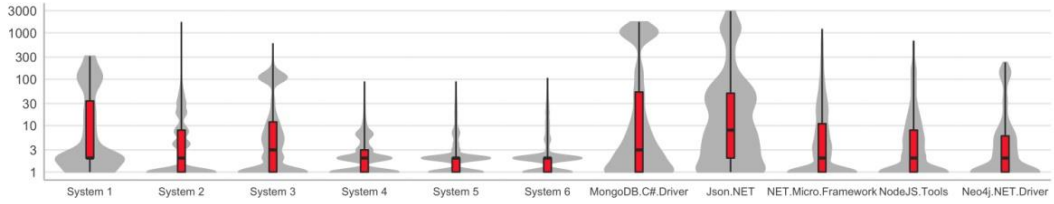

Рис. 6. Функция плотности для FCYC для всех систем
Рис. 6. Density function for FCYC from all systems

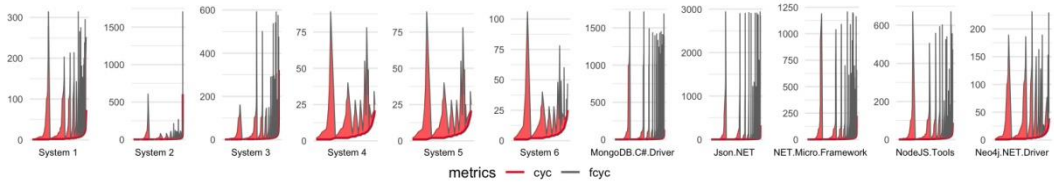

Рис. 7. Расстояние между значениями FCYC и CYC для ранжированных методов всех систем
Fig. 7. Distance between FCYC and CYC values for the ranked methods from all systems

6.4 Вычисление показателя жесткости методов (RNOM)

RNOM — количество методов в подграфе жесткости (или рискованности) данного метода.

6.4.1 Определение

Пусть $\boldsymbol{v_i} \in \boldsymbol{V}$ – метод, $\boldsymbol{\mu = RNOM}$, и $\boldsymbol{ic_i} = \{\boldsymbol{v_1} \cdots \boldsymbol{v_n}\}$ множество методов в $\boldsymbol{ic_i^r}$ (то есть подграф жесткости косвенных связей метода $\boldsymbol{v_i}$), значит:

$$m_i = \mu_i = \mathrm{RNOM}_i = \sum_{k=1}^{n} \mathrm{NOM}_i \qquad (7)$$

6.4.2 Теоретические основы

Метрика RNOM соответствует расширению метрики сложности FAN-IN для некоторого метода. По сути, эта метрика есть результат применения метрики FAN-IN к подграфу жесткости косвенных связей этого метода. Тем самым вычисляется количество методов, прямо или косвенно вызывающих метод, включая сам метод. В этом показателе также учитывается влияние клиентов, подвергшихся рефакторингу при проектировании, на возможности управления размером программ и их сложностью, а также на повышение возможностей повторного использования.

6.4.3 Аналитические оценки

Пусть $\boldsymbol{v_i, v_j} \in \boldsymbol{V}$ два разных произвольных метода в системе $\boldsymbol{E}$. Предполагается, что переменные *FAN-IN*$_i$ и *FAN-IN*$_j$ соответствуют определению i.i.d. (см. Предположение 1), то же самое касается переменных $\boldsymbol{m_i} = \mathrm{RNOM}_i$ и $\boldsymbol{m_j} = \mathrm{RNOM}_j$. Соответственно, с ненулевой вероятностью $\boldsymbol{\mu_i \neq \mu_j}$. Свойство 1 выполняется. Свойство 2 также выполняется (см. раздел 4). С ненулевой вероятностью $\exists\, \boldsymbol{v_k} \in \boldsymbol{V} \mid \boldsymbol{\mu_i = \mu_k}$, значит Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет количество элементов в его подграфе жесткости, поскольку это проектное решение, не зависящее от этой функциональности, следовательно, Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i = m_i}$ и $\boldsymbol{\mu_j = m_j}$, тогда $\boldsymbol{\mu(v_i \oplus v_j) = m_i + m_j - \delta}$, где $\boldsymbol{\delta}$ количество общих методов в графах $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Следовательно, $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_i}$ и $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu_j}$, и Свойство 5 выполняется. Далее, пусть $\boldsymbol{\mu_i = \mu_j}$, и $\exists\, \boldsymbol{v_k} \in \boldsymbol{V}$ такой, что в граф $\boldsymbol{ic_k^r}$

входит $\boldsymbol{\beta}$ методов, общих с графом $\boldsymbol{ic_i^r}$ (см. Предположение 3), и $\boldsymbol{\delta}$ методов, общих с графом $\boldsymbol{ic_j^r}$, причем $\boldsymbol{\beta \neq \delta}$. Тогда,

$$\boldsymbol{\mu(v_i \oplus v_k) = \mu_i + \mu_k - \beta}$$
$$\boldsymbol{\mu(v_j \oplus v_k) = \mu_j + \mu_k - \delta.}$$

Следовательно, $\boldsymbol{\mu(v_i \oplus v_k) \neq \mu(v_j \oplus v_k)}$, и Свойство 6 выполняется. Свойства 7 и 8 также выполняются (см. раздел 4). Для двух методов $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$ пусть $\boldsymbol{\gamma}$ будет количеством методов, общих для графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$, и пусть $\boldsymbol{\delta}$ будет количеством новых методов, добавленных в программу при реструктуризации двух графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$ в новый граф $\boldsymbol{ic_{i'}^r}$. Если $\boldsymbol{\delta > \gamma}$, то ясно, что $\boldsymbol{m_i + m_j - \gamma + \delta > m_i + m_j}$ (то есть $\boldsymbol{\mu(v_i \oplus v_j) > \mu(v_i) + \mu(v_j)}$ при заданных $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$). Следовательно, Свойство 9 выполняется.

6.4.4 Важные соображения

В отношении метрики RNOM можно высказать следующие соображения:

- Метрика RNOM более точно отражает сложность методов, чем показатель FAN-IN. В определенном смысле RNOM –это рекурсивный показатель FAN-IN.
- RNOM – это количество методов, косвенным образом решающих задачу. Эта метрика показывает, сколько времени и усилий потребуется для реализации, понимания (включая отслеживание цепочек вызовов методов) и развития фрагментов программ, которые используют метод, решающий задачу.
- Внесение изменений в метод с бóльшим значением RNOM с большей вероятностью повлияет на необходимость изменений в любом из зависимых методов. Подобный волновой эффект умножает заботы об исправлении ошибок, внесении изменении, проведении тестирования и рефакторинге в методы подграфа жесткости.
- Методы с большим числом зависимых методов, в большей степени считаются подходящими для повторного использования и менее специфичны для приложений.

6.4.5 Экспериментальные данные

Функции плотности метрики RNOM для всех систем показаны на рис. 8, а на рис. 9 представлены линейные диаграммы значений метрик RNOM и FAN-IN для методов во всех системах, ранжированных по FAN-IN.

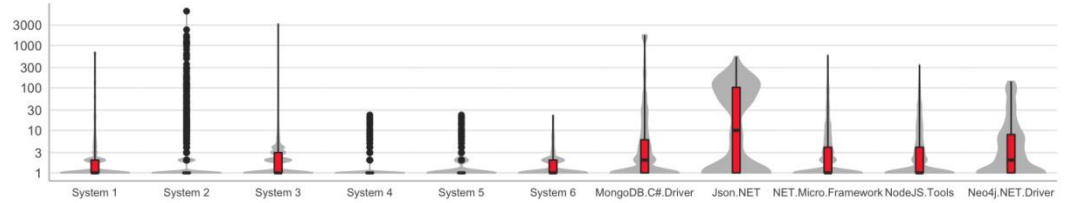

Рис. 8. Функция плотности для RNOM из всех систем
Рис. 8. Density function for RNOM from all systems

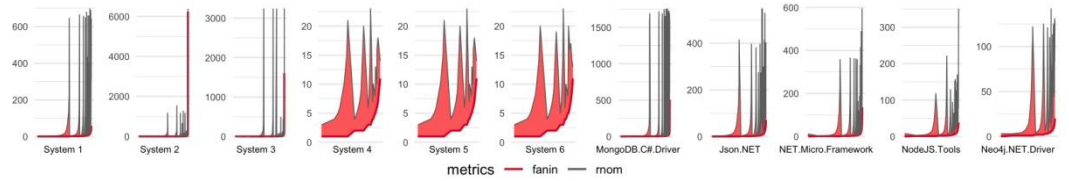

Рис. 9. Расстояние между значениями RNOM и FAN-IN для ранжированных методов всех систем
Fig. 9. Distance between RNOM and FAN-IN values for the ranked methods from all systems

Сводная статистика по метрике RNOM для всех систем представлена в табл. 6. В ней указаны минимум, 1-й квартиль, медиана, 3-й квартиль и максимум для RNOM. Она также показывает наиболее часто отмеченное расстояние MFD между значениями метрик FAN-IN и RNOM и долю методов MFD, имеющих именно такое расстояние MFD, среди всех методов. В ней также представлен коэффициент корреляции Кендалла между значениями метрик FAN-IN и RNOM, а также *p-значение*. Средние значения метрики RNOM для всех конечных методов всех систем приведены в табл. 2.

Табл. 6. Сводная статистика для метрики RNOM
Table 6. Summary Statistics for the RNOM metric

Sys	MFD*	%MDF	$\boldsymbol{\tau}^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
$\boldsymbol{I_1}$	1	84	0.94	0	1	1	1	2	697
$\boldsymbol{I_2}$	1	95	0.98	0	1	1	1	1	6,386
$\boldsymbol{I_3}$	1	76	0.9	0	1	1	1	3	3,238
$\boldsymbol{I_4}$	1	98	0.99	0	1	1	1	1	23
$\boldsymbol{I_5}$	1	93	0.98	0	1	1	1	1	23
$\boldsymbol{I_6}$	1	91	0.97	0	1	1	1	2	23
$\boldsymbol{O_1}$	1	68	0.83	0	1	1	2	6	1,756
$\boldsymbol{O_2}$	1	44	0.68	0	1	1	10	103	548
$\boldsymbol{O_3}$	1	68	0.86	0	1	1	1	4	594
$\boldsymbol{O_4}$	1	71	0.87	0	1	1	1	4	351
$\boldsymbol{O_5}$	1	57	0.76	0	1	1	2	8	142

**MFD: наиболее частое расстояние между RNOM и FAN-IN (MFD: most frequent distance between RNOM and FAN-IN)*

†τ: коэффициент корреляции Кендалла между RNOM и FAN-IN (τ: Kendall correlation coefficient between RNOM and FAN-IN)

6.5 Вычисление показателя жесткости числа строк текста (RLOC)

RLOC – это количество строк кода всех методов в подграфе жесткости данного метода.

6.5.1 Определение

Пусть $\boldsymbol{v_i} \in \boldsymbol{V}$ – метод, $\boldsymbol{\mu} = \mathbf{RLOC}$, и $\boldsymbol{ic_i} = \{\boldsymbol{v_1} \cdots \boldsymbol{v_n}\}$ множество всех методов в графе $\boldsymbol{ic_i^r}$, тогда:

$$m_i = \mu_i = \mathrm{RLOC}_i = \sum_{k=1}^{n} \mathrm{LOC}_k \tag{8}$$

6.5.2 Теоретические основы

RLOC соответствует расширению метрики размера отдельных методов LOC. RLOC – это метрика LOC в применении к подграфу косвенной связи жесткости этого метода (то есть она соответствует суммарному количеству строк кода во всех методах, которые могут прямо или косвенно вызывать метод, а также строк самого этого метода) . В этом показателе также учитывается влияние клиентов, подвергшихся рефакторингу при проектировании, на возможности управления размером программ и их сложностью, а также на увеличение возможностей повторного использования.

6.5.3 Аналитические оценки

Пусть $\boldsymbol{v_i}, \boldsymbol{v_j} \in \boldsymbol{V}$ два разных произвольных метода системы $\boldsymbol{E}$. Переменные $\boldsymbol{LOC_i}$ и $\mathbf{LOC_j}$ предполагаются обладающими свойством i.i.d. (см. Предположение 1), то же самое касается

переменных $\boldsymbol{m_i} = \text{RLOC}_{\boldsymbol{i}}$ и $\boldsymbol{m_j} = \text{RLOC}_{\boldsymbol{j}}$; тогда, с ненулевой вероятностью $\boldsymbol{\mu_i \neq \mu_j}$; Свойство 1 выполняется. Свойство 2 также выполняется (см. раздел 4). С ненулевой вероятностью $\exists\, \boldsymbol{v_k} \in \boldsymbol{V} \mid \boldsymbol{\mu_i = \mu_k}$, следовательно, Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет общее количество строк кода во всех методах в его подграфе жесткости, поскольку это проектное решение, не зависящее от функциональности, следовательно, Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i = m_i}$ и $\boldsymbol{\mu_j = m_j}$, тогда $\boldsymbol{\mu(v_i \oplus v_j) = m_i + m_j - \delta}$, где $\boldsymbol{\delta}$ – количество строк кода в общих методах графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Тогда, $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu(v_i)}$ и $\boldsymbol{\mu(v_i \oplus v_j) \geq \mu(v_j)}$, и Свойство 5 выполняется. Далее, пусть $\boldsymbol{\mu_i = \mu_j}$, и $\exists\, \boldsymbol{v_k} \in \boldsymbol{V}$ такой, что в граф $\boldsymbol{ic_k^r}$ входит $\boldsymbol{\beta}$ строк кода, общих с графом $\boldsymbol{ic_i^r}$ (см. Предположение 3), и $\boldsymbol{\delta}$ строк кода, общих с графом $\boldsymbol{ic_j^r}$, причем $\boldsymbol{\beta \neq \delta}$. Тогда,

$$\boldsymbol{\mu(v_i \oplus v_k) = \mu_i + \mu_k - \beta}$$
$$\boldsymbol{\mu(v_j \oplus v_k) = \mu_j + \mu_k - \delta}.$$

Следовательно, $\boldsymbol{\mu(v_i \oplus v_k) \neq \mu(v_j \oplus v_k)}$, и Свойство 6 выполняется. Свойства 7 и 8 также выполняются (см. раздел 4). Для любых двух методов $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$, пусть $\boldsymbol{\gamma}$ будет значением метрики LOC методов, общих для графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$, и пусть $\boldsymbol{\delta}$ будет значением метрики LOC новых программ, написанных при реструктуризации графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$ в новый граф $\boldsymbol{ic_{i'}^r}$. Если $\boldsymbol{\delta > \gamma}$, становится истинным отношение $\boldsymbol{m_i + m_j - \gamma + \delta > m_i + m_j}$ (то есть $\boldsymbol{\mu(v_i \oplus v_j) > \mu(v_i) + \mu(v_j)}$ при заданных $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$). Следовательно, Свойство 9 выполняется.

6.5.4 Важные соображения

Ниже приводится список перспектив, рассматриваемых в отношении RLOC:

- RLOC – это расширение LOC как метрики размера отдельного метода (то есть RLOC – это рекуррентное накопление LOC).
- Совокупный размер всех методов, которые косвенно зависят от некоторой нужной всем им функциональности, также может помочь спрогнозировать время и усилия, необходимые для разработки, понимания (включая локализацию зависимостей) и изменения требований, которые используют общий для всех метод.
- Более высокие значения RLOC некоторого метода увеличивают вероятность того, что изменение в нем затронет любую из зависящих от него частей программы (возможно возникновение каскадов ошибок или изменений, потребности в тестировании, реструктуризации).
- Считается, что методы с большим количеством зависимых методов легче использовать повторно; они менее специфичны для приложения.

6.5.5 Экспериментальные данные

Статистика метрики RLOC по всем системам представлена в табл. 7. В этой таблице показаны минимум, 1-й квартиль, медиана, 3-й квартиль и максимум для RLOC. Она также показывает наиболее часто отмеченное расстояние MFD между значениями метрик LOC и RLOC и долю методов MFD, имеющих именно такое расстояние MFD, среди всех методов. В ней также представлен коэффициент корреляции Кендалла между значениями метрик LOC и RLOC, а также *p-значение*.

Средние значения метрики RLOC для всех конечных методов всех систем показаны в табл. 2, а функции плотности метрики RLOC для всех систем изображены на рис. 10. Линейные диаграммы значений метрик RLOC и LOC для методов во всех системах, ранжированных по LOC, представлены на рис. 11.

Табл. 7. Сводная статистика для метрики RLOC
Table 7. Summary Statistics for the RLOC metric

Sys	MFD*	%MDF	$\tau^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
I_1	0	62	0.59	0	1	2	5	21	5,608
I_2	0	75	0.68	0	1	1	4	12	140,705
I_3	0	52	0.53	0	1	2	8	27	75,327
I_4	0	94	0.94	0	1	1	6	12	196
I_5	0	77	0.82	0	1	1	4	8	196
I_6	0	68	0.75	0	1	2	3	8	196
O_1	0	47	0.44	0	1	2	6	44	12,370
O_2	0	37	0.28	0	1	3	87	1,306	4,235
O_3	0	52	0.46	0	1	1	9	65	9,181
O_4	0	55	0.45	0	1	2	7	41	3,956
O_5	0	40	0.28	0	1	2	10	49	724

**MFD: наиболее частое расстояние между RLOC и LOC (MFD: most frequent distance between RLOC and LOC)*
$^{\dagger}\tau$: коэффициент корреляции Кендалла между RLOC и LOC (τ: Kendall correlation coefficient between RLOC and LOC)

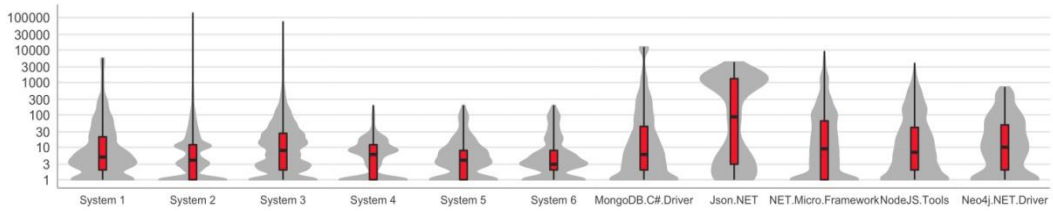

Рис. 10. Функция плотности для RLOC из всех систем
Fig. 10. Density function for RLOC from all systems

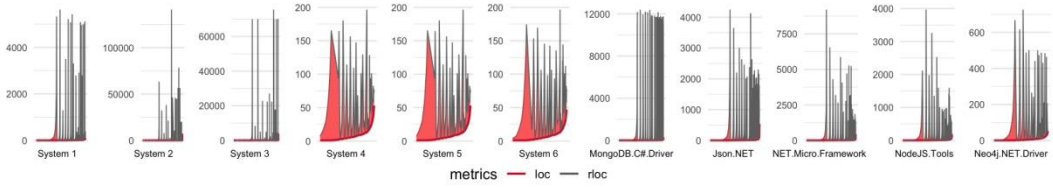

Рис. 11. Расстояние между RLOC и LOC для ранжированных методов всех систем
Fig. 11. Distance between RLOC and LOC values for the ranked methods from all systems

6.6 Вычисление показателя жесткости цикломатической сложности (RCYC)

RCYC – сумма значений сложности всех методов подграфа жесткости данного метода.

6.6.1 Определение

Пусть $\boldsymbol{v_i} \in \boldsymbol{V}$ – метод, $\boldsymbol{\mu = RCYC}$, и $\boldsymbol{ic_i} = \{\boldsymbol{v_1} \cdots \boldsymbol{v_n}\}$ – множество методов в графе $\boldsymbol{ic_i^r}$, тогда:

$$m_i = \mu_i = \mathrm{RCYC}_i = \sum_{k=1}^{n} \mathrm{CYC}_k - (n-1) \qquad (9)$$

В формуле (9) вычитаемое значение $(n-1)$ нужно для тех же целей, что и при вычислении метрики FCYC.

6.6.2 Теоретические основы

RCYC соответствует расширению метрики сложности CYC для метода. RCYC – это метрика CYC, примененная к подграфу косвенной связи жесткости этого метода (то есть она соответствует совокупной цикломатической сложности всех методов, которые могут прямо или косвенно вызывать метод, включая сложность самого этого метода). Вычитаемое в формуле (9) компенсирует лишние $(n-1)$ единиц. В этом показателе также учитывается влияние клиентов, подвергшихся рефакторингу при проектировании, на возможности управления размером программ и их сложностью, а также на повышение возможностей повторного использования.

6.6.3 Аналитические оценки

Пусть $\boldsymbol{v_i, v_j} \in \boldsymbol{V}$ – любые два разных метода системы $\boldsymbol{E}$. Предполагается, что переменные $\boldsymbol{CYC_i}$ и $\mathbf{CYC_j}$ обладают свойствами i.i.d. (см. Предположение 1), это же предположение относится к переменным $\boldsymbol{m_i} = \mathrm{RCY}\mathbf{C_i}$ и $\boldsymbol{m_j} = \mathrm{RCY}\mathbf{C_j}$. Тогда с ненулевой вероятностью $\boldsymbol{\mu_i} \neq \boldsymbol{\mu_j}$; Свойство 1 выполняется. Свойство 2 также выполняется (см. раздел 4).

Существует ненулевая вероятность того, что $\exists\ \boldsymbol{v_k} \in \boldsymbol{V} \mid \boldsymbol{\mu_i} = \boldsymbol{\mu_k}$, следовательно, Свойство 3 выполняется. Функциональность, выполняемая методом, не определяет цикломатическую сложность методов в его подграфе жесткости, поскольку это проектное решение, от функциональности не зависящее, Свойство 4 выполняется. Пусть $\boldsymbol{\mu_i} = \boldsymbol{m_i}$ и $\boldsymbol{\mu_j} = \boldsymbol{m_j}$, тогда $\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_j}) = \boldsymbol{m_i} + \boldsymbol{m_j} - \boldsymbol{\delta}$, где $\boldsymbol{\delta}$ – значение цикломатической сложности для методов, общих в графах $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$. Максимальное значение $\boldsymbol{\delta}$ равно $\boldsymbol{\min(m_i, m_j)}$. Следовательно, $\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_j}) \geq \boldsymbol{\mu_i}$ и $\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_j}) \geq \boldsymbol{\mu_j}$, и Свойство 5 выполняется. Далее, пусть $\boldsymbol{\mu_i} = \boldsymbol{\mu_j}$, и $\exists\ \boldsymbol{v_k} \in \boldsymbol{V}$ такой, что цикломатическая сложность методов, общих в графах $\boldsymbol{ic_k^r}$ и $\boldsymbol{ic_i^r}$, равна $\boldsymbol{\beta}$ (см. Предположение 3), а цикломатическая сложность методов, общих в графах $\boldsymbol{ic_k^r}$ и $\boldsymbol{ic_j^r}$, равна $\boldsymbol{\delta}$, причем $\boldsymbol{\beta} \neq \boldsymbol{\delta}$. Тогда,

$$\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_k}) = \boldsymbol{\mu_i} + \boldsymbol{\mu_k} - \boldsymbol{\beta}$$
$$\boldsymbol{\mu}(\boldsymbol{v_j} \oplus \boldsymbol{v_k}) = \boldsymbol{\mu_j} + \boldsymbol{\mu_k} - \boldsymbol{\delta}.$$

Таким образом, $\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_k}) \neq \boldsymbol{\mu}(\boldsymbol{v_j} \oplus \boldsymbol{v_k})$. Свойство 6 выполняется. Свойства 7 и 8 также выполняются (см. раздел 4). Для двух произвольных методов $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$, пусть $\boldsymbol{\gamma}$ будет значением CYC для методов, общих в графах $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$, и пусть $\boldsymbol{\delta}$ будет значением CYC новых программ, написанных для объединения графов $\boldsymbol{ic_i^r}$ и $\boldsymbol{ic_j^r}$ в новый граф $\boldsymbol{ic_i^r}$. Если $\boldsymbol{\delta} > \boldsymbol{\gamma}$, то $\boldsymbol{m_i} + \boldsymbol{m_j} - \boldsymbol{\gamma} + \boldsymbol{\delta} > \boldsymbol{m_i} + \boldsymbol{m_j}$ (то есть, $\boldsymbol{\mu}(\boldsymbol{v_i} \oplus \boldsymbol{v_j}) > \boldsymbol{\mu_i} + \boldsymbol{\mu_j}$ при заданных $\boldsymbol{v_i}$ и $\boldsymbol{v_j}$). Следовательно, Свойство 9 выполняется.

6.6.4 Важные соображения

В отношении метрики RCYC можно высказать следующие соображения:

- RCYC – более точная метрика цикломатической сложности метода, чем CYC, поскольку она учитывает значения CYC методов-клиентов.
- Объединенная цикломатическая сложность всех методов, которые должны выполнять какую-то общую работу, также может позволить прогнозировать время и усилия, необходимые для программирования, понимания (включая обнаружение подграфа жесткости или рискованности) и сопровождения функций, в реализации которых участвует данный метод.
- Методы с бóльшими значениями метрики RCYC могут с большей вероятностью распространить влияние любых изменений на вызывающие их методы.
- Методы с более сложными зависимостями легче использовать повторно, они менее специфичны для приложения.

6.6.5 Экспериментальные данные

Сводная статистика по метрике RCYC во всех системах представлена в табл. 8. В этой таблице показаны минимум, 1-й квартиль, медиана, 3-й квартиль и максимум для RCYC. Она также показывает наиболее часто отмеченное расстояние MFD между значениями метрик CYC и RCYC и долю методов MFD, имеющих именно такое расстояние MFD, среди всех методов. В ней также представлен коэффициент корреляции Кендалла между значениями метрик CYC и RCYC, а также *p-значение*.

Табл. 8. Сводная статистика для метрики RCYC
Table 8. Summary Statistics for the RCYC metric

Sys	MFD*	%MDF	$\tau^{\dagger}$	*p-val*	Min	1st Q	Med	3rd Q	Max
I_1	0	73	0.61	0	1	1	2	7	1,517
I_2	0	82	0.74	0	1	1	2	4	29,226
I_3	0	58	0.59	0	1	1	2	6	11,872
I_4	0	97	0.95	0	1	1	2	3	77
I_5	0	90	0.83	0	1	1	2	2	77
I_6	0	85	0.76	0	1	1	2	2	77
O_1	0	60	0.48	0	1	1	2	14	3,620
O_2	0	42	0.31	0	1	1	48	733	2,193
O_3	0	57	0.47	0	1	1	3	22	3,362
O_4	0	59	0.5	0	1	1	3	18	1,674
O_5	0	49	0.3	0	1	1	4	23	273

**MFD: наиболее частое расстояние между RCYC и CYC (MFD: most frequent distance between RCYC and CYC)*

†τ: коэффициент корреляции Кендалла между RCYC и CYC (τ: Kendall correlation coefficient between RCYC and CYC)

Средние значения метрики RCYC для всех конечных методов всех систем приведены в табл. 2. Функции плотности метрики RCYC для всех систем показаны на рис. 12. Линейные диаграммы значений метрик RCYC и CYC для методов во всех системах, ранжированные по CYC, представлены на рис. 13.

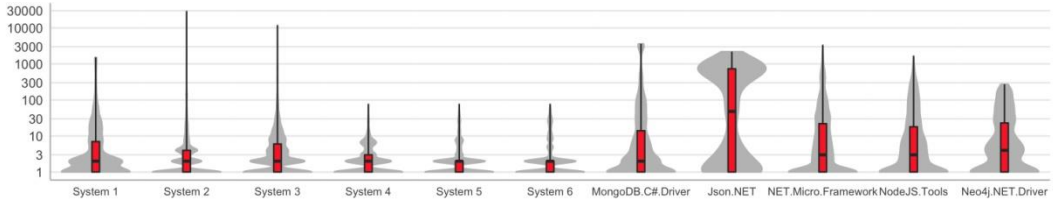

Рис. 12. Функция плотности для RCYC из всех систем
Рис. 12. Density function for RCYC from all systems

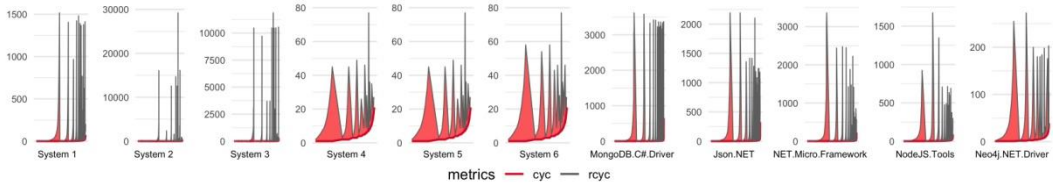

Рис. 13. Расстояние между RCYC и CYC для ранжированных методов всех систем
Fig. 13. Distance between RCYC and CYC values for the ranked methods from all systems

6.7 Обсуждение

В этом разделе обсуждаются рейтинги, присваиваемые метриками, аналитические результаты и результаты, имеющие отношение к управлению проектами.

6.7.1 Ранжирование с помощью метрик

Шесть представленных показателей косвенной связи измеряют различные и независимые атрибуты размера или сложности, используя либо хрупкость (неустойчивость) связи, либо ее жесткость (рискованность). Существуют три метрики хрупкости, основанные на одних и тех же подграфах хрупкости каждого метода, а также три метрики жесткости, основанные на одних и тех же подграфах жесткости. Нам могут возразить, заявив, что достаточно иметь только одну метрику хрупкости и одну метрику жесткости. В этой статье для подтверждения разницы между показателями использовался коэффициент корреляции Кендалла. В табл. 9 представлены коэффициенты для каждой системы и для каждой пары связанных показателей.

Табл. 9. Коэффициент корреляции Кендалла τ для каждой пары показателей
Table 9. Kendall's τ correlation coefficient for every pair of metrics

Sys	FNOM FLOC	FNOM FCYC	FLOC FCYC	RNOM RLOC	RNOM RCYC	RLOC RCYC
I_1	0.79	0.79	0.81	0.66	0.67	0.81
I_2	0.77	0.8	0.91	0.36	0.27	0.81
I_3	0.82	0.7	0.83	0.54	0.55	0.75
I_4	0.13	0.16	0.85	0.18	0.2	0.86
I_5	0.36	0.39	0.7	0.45	0.49	0.76
I_6	0.51	0.48	0.6	0.62	0.63	0.71
O_1	0.85	0.76	0.84	0.78	0.68	0.77
O_2	0.85	0.81	0.91	0.88	0.88	0.93
O_3	0.78	0.73	0.85	0.74	0.72	0.87
O_4	0.73	0.7	0.82	0.75	0.74	0.87
O_5	0.81	0.74	0.84	0.83	0.76	0.85

Метрики FLOC и RLOC являются парными метриками косвенных связей, связанными с атрибутами размера методов в подграфах хрупкости и жесткости. Это значит, что метрики FLOC и RLOC можно оставить в наборе метрик, поскольку они измеряют размер методов, а не их сложность. Кроме того, больший размер кода не означает более сложный код и наоборот. FLOC имеет значение корреляции τ, находящееся в диапазоне от 0,13 до 0,85 для FNOM, и значение τ, находящееся в диапазоне от 0,60 до 0,91 для FCYC. Аналогично, RLOC имеет τ от 0,18 до 0,88 для RNOM и τ от 0,71 до 0,93 для RCYC. Более высокие значения τ указывают на то, что соответствующая пара метрик присваивает методам в системе почти одинаковый рейтинг, но поскольку корреляция не идеальна, не все методы получают одинаковый рейтинг по обеим метрикам. С помощью FLOC и RLOC можно найти наиболее хрупкие или наиболее жесткие методы, рассматривая размеры косвенно связанных с ними методов.

6.7.2 Обсуждение аналитических результатов

Как показано в разделе 4, все предложенные метрики соответствуют расчетным критериям. Выявленные отношения между хрупкостью и жесткостью косвенных связей позволяют создавать робастные метрики, измеряющие размер и сложность программных систем.

6.7.3 Обсуждение результатов по управлению проектами

Администраторы и технические руководители процессами проектирования могут использовать набор метрик косвенных связей для оценки и управления размерами и сложностью программных проектов. Метрика FNOM может выявлять методы со многочисленными косвенными зависимостями. Вероятно, что именно эти методы реализуют наиболее сложные системные функции. Чтобы создать правильный проект и правильно его реализовать, именно этим методам следует уделять особое внимание,

Некоторая функциональность может приводить к существенному числу зависимостей, но метрика FNOM не сумеет этого выявить, однако, метрика FLOC обнаружит, что они реализованы многими и многими строками кода. Размер, измеряемый суммирующей метрикой LOC, учитывающей зависимости некоторого метода, поможет обнаружить такую функциональность и определить, не нужно ли разделить некоторые крупные фрагменты на более мелкие части, или провести рефакторинг программ. Какие-то другие функции могут быть рассредоточены по множеству длинных зависимостей, их можно обнаружить с помощью метрик FNOM и FLOC.

Метрика FCYC может помочь обнаружить функциональность, граф управления которой состоит из множества линейно независимых путей. Такое знание может оказаться полезным при разработке тестовых примеров для проверки функциональности и принятии решения о необходимости рефакторинга. Информация в графе ICG_i^f метода v_i также дает представление о способах реализации его функциональной структуры.

Остальные три метрики жесткости позволяют анализировать и обнаруживать атрибуты размера и сложности программ с учетом зависимых или повторно используемых методов. Метрика RNOM позволяет узнать количество методов, косвенно зависящих от данного метода, и дает фактическую степень повторного использования этого многим нужного метода. Благодаря этому пониманию специалист по сопровождению может изучить все зависимые методы и определить, какие из них необходимо изменить при корректировке переиспользуемого метода. Эти зависимые методы должны подвергнуться дополнительному тестированию. Если значение RNOM становится слишком большим для какого-либо метода v_i, специалист по сопровождению программ может проверить его подграф жесткости ICG_i^r и определить необходимость рефакторинга.

Принципы модульности и повторного использования широко используются разработчиками программного обеспечения для борьбы со сложностью программ. Следовательно, переиспользуемые программные компоненты встречаются нередко. Изменение такого рода методов чревато побочными эффектами в каждой строке кода, которая косвенно их переиспользует. И это кроме тех дополнительных усилий, которые необходимы для понимания того, какой из этих зависимых методов будет затронут изменениями. В результате могут стать необходимыми дополнительные изменения в зависимых методах, внедрение в программы новых скрытых ошибок или затруднение с обнаружением старых, рост потребностей в дополнительных ресурсах на проведение тестирования, или, в самом худшем случае, полный рефакторинг связанных методов. Метрика жесткости RLOC позволяет оценить подобные потенциальные последствия в строках кода. Объединив метрики RLOC и RNOM, можно обнаружить те методы и их подграфы ICG^r, которые имеют большое количество методов-клиентов, большой размер кода в строках кода методов-клиентов или и то, и другое одновременно.

Наконец, метрика RCYC может помочь измерить количество линейно независимых путей, на которых переиспользуется некоторый служебный метод. Эта метрика даст более точную оценку усилий, необходимых для тестирования метода после его изменения, или убедит в необходимости рефакторинга из-за технических недоработок.

6.8 Угрозы достоверности

Конструктивная достоверность. Проблема конструктивной достоверности связана с тем, действительно ли измеряем то, что хотим. Предлагаемый набор метрик построен на основе хорошо известных по литературе метрик: FAN-IN, FAN-OUT, LOC и CYC. Каждая метрика вычисляется суммированием метрик хрупкости и жесткости методов вдоль ребер графа косвенных связей ICG. Подграфы хрупкости отражают синтаксическую структуру разделенной на части и инкапсулированной функциональности, тогда как подграфы жесткости имеют дело со статическими шаблонами переиспользования. Обе статические структуры кода воплощают атрибуты размера и сложности, связанные с реализуемой ими функциональностью, и являются результатом хорошей практики проектирования, направленной на управление сложностью. Чтобы снизить вероятность того, что метрики хрупкости и жесткости не будут измерять ничего, кроме собственных базовых показателей (то есть FAN-OUT для FNOM, LOC для FLOC и так далее), к каждой паре базовых и косвенных метрик связи мы применили корреляционный анализ Кендалла. Каждая из шести возможных пар метрик хрупкости и жесткости (то есть FNOM-FLOC, FNOM-FCYC и FLOC-FCYC, RNOM-RLOC, RNOM-RCYC и RLOC-RCYC) подверглась одинаковому анализу. Это дало гарантию, что информация, извлеченная каждой из метрик, не является избыточной. Был также проведен анализ значений метрик прямых и косвенных связей с использованием линейных диаграмм этих значений для методов, ранжированных по значению прямой метрики, а затем по значению метрики косвенных связей. Это должно показать, что введенные метрики показывают результаты, отличные от значений соответствующих базовых показателей.

Внутренняя достоверность. В данном исследовании предполагается, что каждая из базовых метрик, применяемых к каждому методу в системе, представляет собой независимую и одинаково распределенную дискретную случайную величину (то есть нельзя предсказать значения метрик одного метода на основе метрик других методов). Мы предполагаем то же самое в отношении метрик косвенных связей. Чтобы гарантировать справедливость этого предположения, мы полагаемся на тот факт, что структура каждого подграфа косвенных связей вытекает из решений проектировщиков и разработчиков программного обеспечения, а не из значений метрик. Мы также ссылаемся на то, что метрики удовлетворяют всем критериям достоверности, предложенным в работе [34].

Внешняя достоверность. Внешняя достоверность относится к обобщению результатов исследования. Мы изучили шесть промышленных систем от двух разных компаний, две из них были разработаны по классической транзакционной схеме “Модель-Представление-Контроллер” (Model-View-Controller, MVC), три – на основе самостоятельно созданной инфраструктуры быстрой разработки приложений (Rapid Application Development, RAD), которая сама рассматривается в качестве системы № 6. Мы также изучили пять систем с открытым исходным кодом. Языком программирования этих систем является C#, что возможно оставляет в стороне системы, написанные с использованием других языков программирования. Кроме того, можно было пропустить некоторые косвенные связи в системе из-за ошибок при анализе исходных текстов. Мы снижаем эти риски, подсчитывая значения метрик с помощью официального компилятора, как это будет делаться с любым другим языком, который будет использован в будущем. Более того, косвенные связи между методами, функциями или модулями в большинстве современных языков программирования аналогичны и служат той же цели.

7. Заключение

В этой статье не только представлены метрики, которые имеют прочную теоретическую основу и были тщательно изучены, но эти метрики также применены к исследованию нескольких реальных коммерческих проектов, использовавших язык C#, и к программным

системам с открытым исходным кодом. В статье представлена полезная информация о том, как эти метрики можно использовать при сопровождении программного обеспечения, а также предлагаются рекомендации по их эффективной интеграции. Далее следует некоторый краткий обзор этих рекомендаций.

Опытный, квалифицированный руководитель проекта для оценки времени и усилий, необходимых для понимания и сопровождения функциональности методов объектов может использовать метрику FNOM. Эту информацию можно использовать для сравнения требований к ресурсам различных методов и систем. Кроме того, FNOM может выявлять методы, которые с набольшей вероятностью подвержены влиянию ошибок и изменений в связанных с ними программах. Это поможет своевременно выделить ресурсы для тестирования их функциональности после проведения модификаций. Благодаря собранной информации руководитель проекта может принимать обоснованные решения и оптимизировать распределение ресурсов, обеспечивая эффективное и результативное проведение проекта.

Помимо оценки технических недоработок, этот подход может помочь принять решение о необходимости проведения рефакторинга программных систем.

Также могут помочь достижению тех же целей метрики FLOC и FCYC, одним и тем же методам системы они дают оценки, отличающиеся от оценок FNOM. Эти метрики могут выявить более крупные или более сложные процедуры с точки зрения количества строк кода или цикломатической сложности, а это может помочь принять решения о приоритетах в рефакторинге.

Для оценки хрупкости системы рекомендуется использовать все метрики в совокупности. Сравнивая среднюю хрупкость методов разных систем, руководство проектов может принимать обоснованные решения о перераспределении ресурсов между различными системами. Высокий уровень хрупкости может также указывать на ограниченную возможность повторного использования кода, поскольку более сложный и специфичный код имеет тенденцию быть менее пригодным для повторного использования. Более того, подграф хрупкости методов, которые необходимо изменить, может проявить функциональность, которая требует поддержки при сопровождении.

Метрики жесткости (рискованности) в первую очередь ориентированы на измерение возможности повторного использования, размера и сложности подграфов жесткости методов. Каждая из этих метрик – RNOM, RLOC и RCYC – обеспечивает различное ранжирование методов, их можно комбинировать и определять, какие из них имеют зависимый код, который либо более обширен, либо более сложен, чем другие. Особенно жесткие методы (имеющие высокий риск зависимости от обращающихся к ним методов), выявленные с помощью этих показателей, требуют большего внимания с точки зрения обновления и тестирования по сравнению с менее жесткими методами (с низким уровнем такого риска). Разработчики могут использовать эти метрики для определения приоритетности усилий по сопровождению и тестированию, гарантируя, что высоко рискованные методы особенно тщательно модифицируются и тестируются перед включением в работу. Используя метрики жесткости, команды разработчиков программного обеспечения могут заранее выявлять и устранять проблемы, влияющие на качество и сопровождение программного обеспечения.

Список литературы / References

[1]. M. M. Lehman, J. F. Ramil, P. D. Wernick, D. E. Perry, and W. M. Turski, “Metrics and laws of software evolution-the nineties view”, Proceedings Fourth International Software Metrics Symposium (1997) pp. 20–32.

[2]. Almeyda and A. Dávila, “Process improvement in software requirements engineering: A systematic mapping study”, Programming and Computer Software 48, 513–533 (2022).

[3]. ISO/IEC 14764, Software Engineering – Software Life Cycle Processes – Maintenance, Standard (International Organization for Standardization, 2006(E)).
[4]. Priyadarshi Tripathy Naik and Kshirasagar, Software Evolution and Maintenance: A Practioner's Approach (John Wiley & Sons, Inc, 2015) p. 393.
[5]. Shyam R. Chidamber and Chris F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", in OOPSLA '91 Conference proceedings on Object-oriented programming systems, languages, and applications (ACM Digital Library, Phoenix, Arizona, USA, 1991) pp. 197–211.
[6]. Wei Li and Sallie Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software 23, 111–122 (1993).
[7]. Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering 20, 476–493 (1994).
[8]. Lionel Briand, Prem Devanbu, and Walcelio Melo, "An investigation into coupling measures for C++", in ICSE '97 Proceedings of the 19th international conference on Software (ACM Digital Library, 1997) pp. 412–421.
[9]. Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller, "Mining Version Histories to Guide Software Changes", in Proceedings of the 26th International Conference on Software Engineering (IEEE Computer Society, 2004) pp. 563–572.
[10]. Ewan Tempero and Paul Ralph, "A Framework for Defining Coupling Metrics", Science of Computer Programming, 1–17 (2018).
[11]. Timothy C. Lethbridge and R. Lagani`ere, Object-Oriented Software Engineering: Practical Software Development Using UML and Java, 2nd ed. (McGraw-Hill, 2005) p. 561.
[12]. Hong Yul Yang, Measuring Indirect Coupling, Ph.D. thesis, University of Auckland (2010).
[13]. Shari Lawrence Pfleeger and Shawn Bohner, "A Framework for Software Maintenance Metrics", in Proceedings of the Conference on Software Maintenance (1990) pp. 320–327.
[14]. Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy, "Using information retrieval based coupling measures for impact analysis", Empirical Software Engineering 14, 5–32 (2009).
[15]. Gabriele Bavota, Bogdan Dit, Rocco Oliveto, Massimilano Di Penta, Denys Poshyvanyk, and Andrea De Lucia, "An empirical study on the developers' perception of software coupling", in Proceedings of the International Conference on Software Engineering (ICSE '13) (IEEE, 2013) pp. 692–701.
[16]. A. M. Frolov, "A hybrid approach to enhancing the reliability of software", Programming and Computer Software 30, 18–24 (2004).
[17]. Johann Eder, Gerti Kappel, and Michael Schrefl, Coupling and Cohesion in Object-Oriented Systems, Tech. Rep. 1 (University of Klagenfurt, Austria, 1992).
[18]. M Hitz and B Montazeri, "Measuring coupling and cohesion in object-oriented systems", Proceedings of the International Symposium on Applied Corporate Computing 50, 75–76 (1995).
[19]. Thomas Zimmermann and Nachiappan Nagappan, "Predicting defects using network analysis on dependency graphs", Proceedings of the 30th International Conference on Software Engineering, 531 (2008).
[20]. Nasib S. Gill and Balkishan, "Dependency and interaction oriented complexity metrics of component-based systems", ACM SIGSOFT Software Engineering Notes 33, 1 (2008).
[21]. V. N. Kasyanov, "Graph applications in programming", Programming and Computer Software 27, 146–164 (2001).
[22]. L.C. Briand, J. Wust, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems", in Proceedings IEEE International Conference on Software Maintenance (ICSM '99) (IEEE Xplore, Oxford, England, UK, 1999) pp. 475–482.
[23]. Alan MacCormack, John Rusnak, and Carliss Baldwin, "Exploring the duality between product and organizational architectures: A test of the "mirroring" hypothesis", Research Policy 41, 1309–1324 (2012).
[24]. M. Durán, R. Juárez-Ramírez, S. Jiménez, and C. Tona, "User story estimation based on the complexity decomposition using Bayesian networks", Programming and Computer Software 46, 569–583 (2020).
[25]. F. Valdés-Souto and Lizbeth Naranjo-Albarrán, "Improving the software estimation models based on functional size through validation of the assumptions behind the linear regression and the use of the confidence intervals when the reference database presents a wedge-shape form", Programming and Computer Software 47, 673–693 (2021).
[26]. Huan Li and Bing Li, "A pair of coupling metrics for software networks", Journal of Systems Science and Complexity 24, 51–60 (2011).

[27]. Ran Mo, Yuanfang Cai, Rick Kazman, Lu Xiao, and Qiong Feng, “Decoupling level: A New Metric for Architectural Maintenance Complexity”, Proceedings of the 38th International Conference on Software Engineering - ICSE ’16, 499–510 (2016).
[28]. Saleh Almugrin, Waleed Albattah, and Austin Melton, “Using indirect coupling metrics to predict package maintainability and testability”, Journal of Systems and Software 121, 298–310 (2016).
[29]. Robert Lagerstr¨om, Carliss Baldwin, Alan MacCormack, Dan Sturtevant, and Lee Doolan, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 10379 LNCS (2017) pp. 53–69.
[30]. N. I. V’yukova, V. A. Galatenko, and S. V. Samborskii, “Dynamic program analysis tools in gcc and clang compilers”, Programming and Computer Software 46, 281–296 (2020).
[31]. A. A. Timakov, “Information flow control in software db units based on formal verification”, Programming and Computer Software 48, 265–285 (2022).
[32]. Robert C. Martin, Agile Software Development: Principles, Patterns, and Practices (Pearson Education, Inc., New Jersey, USA, 2003) p. 557.
[33]. Norman Fenton, “Software Measurement: A Necesary Scientific Basis”, IEEE Transactions on Software Engineering 20, 199–206 (1994).
[34]. Elaine J Weyuker, “Evaluating Software Complexity Measures”, IEEE Transactions on Software Engineering 14, 1357–1365 (1988).
[35]. Sriram Pemmaraju and Steven Skiena, Computational discrete mathematics: combinatorics and graph theory with Mathematica (Caambridge University Press, 2003) p. 497.
[36]. Thomas J. McCabe, “A Complexity Measur”, IEEE Transactions on Software Engineering SE-2, 308–320 (1976).
[37]. Harold N. Gabow, “Path-based depth-first search for strong and biconnected components”, Information Processing Letters 74, 107–114 (2000).
[38]. MongoDB, “.NET Driver for MongoDB” (2019).
[39]. Newtonsoft, “Json.NET: Popular high-performance JSON Framework for .NET” (2019).
[40]. NETMF, “.NET Micro Framework Interpreter” (2019).
[41]. Microsoft, “Node.js tools for Visual Studio” (2019).
[42]. Neo4j, “Neo4j .NET Driver” (2019).
[43]. Maurice G. Kendall, Rank Correlation Methods, 4th ed. (Griffin London, London, England, 1970) p. 202.
[44]. All density functions in the Figures use log10 scale to avoid the graphics from squashing.
[45]. Except those methods whose difference between FNOM and FAN-OUT is the MFD (this applies to all other line plots as well).
[46]. M. V. Ksenzov, “Architectural refactoring of corporate program systems”, Programming and Computer Software 32, 31–43 (2006).
[47]. M. H. Halstead, “Toward a theoretical basis for estimating programming effort”, in ACM 1975 Annual Conference (ACM Digital Library, 1975) pp. 222–224.

Информация об авторах / Information about authors

Хосе НАВАС-СУ – бакалавр в области компьютерных наук Технологического института Коста-Рики (1994), магистр в области компьютерных наук из Технологического института Коста-Рики с отличием Magna Cum Laude (2017), кандидат наук в области инженерии в Технологическом институте Коста-Рики под руководством доктора Антонио Гонсалеса Торреса. Работал инженером-программистом в течение трех десятилетий, включая работу в многонациональных компаниях, таких как Accenture и GFT. Преподаватель кафедры компьютерных наук Коста-Риканского технологического института.

Jose NAVAS-SU is an Instructor at the Department of Computer Science of the Costa Rica Institute of Technology. He is a candidate for Ph.D. in Engineering at the Costa Rica Institute of Technology under the supervision of Dr. Antonio González Torres, he obtained a master’s degree in Computer Science (2017) from the Costa Rica Institute of Technology with the Magna Cum Laude distinction, and obtained a bachelor’s degree in Computer Science (1994) from the Costa Rica Institute of Technology. José has worked as Software Engineer for three decades, including working for multinational companies, such as Accenture and GFT.

Антонио ГОНСАЛЕС-ТОРРЕС – бакалавр в области компьютерных наук (1999), магистр в области компьютерных наук (2001) в Университете Коста-Рики, магистр в области интеллектуальных систем (2014) в Университете Саламанки (Испания), доктор философии в области компьютерных наук и автоматизации с отличием Summa Cum Laude, международная докторская степень (2015). В рамках подготовки докторской диссертации выполнял исследования в Открытом университете Великобритании. Доцент кафедры вычислительной техники Коста-Риканского технологического института. Имеет более чем 20-летний профессиональный опыт, в течение которого он работал в нескольких многонациональных компаниях, включая Walmart, Intel, Equifax, Global Exchange group (Eurodivisas) и Sykes. Параллельно с профессиональной работой работал инструктором Cisco и профессором университета, участвовал в нескольких исследовательских проектах. Сфера научных интересов: программная инженерия, разработка методов и инструментов визуальной аналитики и кибербезопасность.

Antonio GONZALEZ-TORRES is an Assistant Professor at the Department of Computer Engineering of the Costa Rica Institute of Technology. He obtained a Ph.D. in Computer Science and Automation with the Summa Cum Laude and the International Doctorate (2015) distinctions, and a master's degree in Intelligent Systems (2014) from the University of Salamanca (Spain). As part of his doctorate, he made a research stay at the Open University of the United Kingdom. In addition, he received a master's degree in Computer Science (2001) and a bachelor's degree in Computer Science (1999) from the University of Costa Rica. Dr. González has more than 20 years of professional experience, during which he has worked for several multinational companies, including Walmart, Intel, Equifax, Global Exchange group (Eurodivisas) and Sykes. In parallel with his professional work, he has worked as a Cisco instructor and university professor and has participated in several research projects. His area of expertise is software engineering, the design of visual analytics methods and techniques, and cybersecurity.