ПЗ 6. Гибридные метрики. Метрика Кокола

(Составители Никонова Г.В. ngvlad@mail.ru, Никонов А.В. nalva@mail.ru)

ЭЛЕКТРОННЫЙ ВАРИАНТ

https://disk.yandex.ru/d/xgBQ5OhfmR1fdg

6.1 ГИБРИДНЫЕ МЕТРИКИ

Невозможно контролировать качество программных продуктов **без числовых показателей. Нужна математическая модель** для определения показателей качества программного кода [2].

На сегодняшний день **существует несколько сотен различных метрик кода**, которые отражают те или иные аспекты программного кода. **Однако использование необработанных значений этих метрик не всегда может быть эффективным**.

Поэтому **требуется наличие системы метрик**, которая **объективно показывала бы различные свойства программного кода**.

Качество программного кода подразумевает грамотно выдержанный **архитектурный стиль** программного кода, чёткое **разделение кода на функциональные блоки** и **строгую структуризацию**, и т. д [3].

Существует большое количество метрик, анализирующих различные аспекты кода.

Для построения объективного представления о программном коде **нужно использовать связанный набор метрик**, которые будут **отражать целостное представление о качестве** программного кода.

Как правило, **качественный код представляется как** программный код, **не наделённый избыточной сложностью и связностью частей системы, хорошо структурированный** и **имеющий адекватные пропорции для объёма** (функциональные блоки не должны быть слишком большими или маленькими, что отражает качество декомпозиции в системе).

Чтобы проанализировать качество программного кода, нужно использовать связанную систему метрик, характеризующую четыре аспекта качества программного кода [4, 5]. **Требуется**

измерить показатели по группам:

– **сложность** программного кода;

– **связность** программного кода;

– **структурированность** программного кода;

– **объёмные характеристики** программного кода.

После вычисления каждой меры в группе, необходимо объединить эти показатели и получить единый результат для каждой группы. Для этого могут быть использованы различные математические методы и подходы.

Одним из наиболее эффективных подходов является **комплексная мера П. Кокола**. Эта **мера берет за основу одну метрику, но также учитывает другие метрики**, которые должны оказывать влияние на конечный результат.

Мера Кокола позволяет получить единственное числовое значение для набора метрик с учётом взвешенных коэффициентов.

1. Первая группа измерений – это **сложность программного кода.** Поскольку при разработке программного кода программисты очень часто занимаются борьбой со сложностью, то **эта группа** будет **одной из важнейших** из представленных выше.

Для измерения сложности обычно применяются метрики сложности Мак-Кейба и его модификации.

Кроме того, для этого **могут использоваться метрики Холстеда**, отражающие трудоёмкость понимания программы и трудоёмкость кодирования. Таким образом можно получить связанную систему метрик для вычисления сложности программного кода.

2. Вторая группа измерений – степень взаимодействия программного кода. Она отражает зависимость отдельных модулей программного кода друг от друга. Как известно, **чем больше таких зависимостей, тем больше сложностей может появляться при модификации и развитии** программной системы.

Основными метриками в этой группе являются метрики внешних (сцепление модулей между собой) и внутренних (связность компонентов внутри модуля) зависимостей – это **центробежные и центростремительные связи**, **глубина наследования** и **среднее число внутренних связей на тип**.

3. Третья группа мер – структурированности программного кода – отражает степень разбиения программного кода на функциональные блоки и уровень инкапсуляции (объединение данных и методов, которые с ними работают, **в один компонент (*например*, класс).**

Чем больше программа структурирована, тем лучше она поддаётся модификации, пониманию и отладке. В хорошо структурированном программном коде повышается возможность повторного использования кода.

В качестве основных мер для этой группы **используются метрика Хансена** и **метрика Пивоварского**. Обе эти **меры являются модификацией меры Мак-Кейба** и **отражают структурированность программы.**

4. Четвёртая группа измерений – это **объёмные характеристики кода**.

Данная **группа** мер **не является прямым показателем качества** программного кода. **Однако значения** данных измерений может быть **опосредованным индикатором структурированности и качества кода.**

Например, эта группа измерений **способна информировать** разработчи-

ков о том, **что какой-либо класс содержит слишком большое или слишком маленькое количество методов и необходимо пересмотреть данный участок кода**.

В качестве основы для измерения данных характеристик используются метрики по модели Холстеда и набор метрик, построенных на основе отношения размеров программных сущностей.

Практика применения различных метрик специалистами программной инженерии **говорит**, что **НЕТ ОДНОЙ УНИВЕРСАЛЬНОЙ МЕТРИКИ**.

Любые **метрические характеристики программы должны контролироваться либо в зависимости друг от друга, либо в зависимости от конкретной задачи** [6].

Рекомендуется применять гибридные меры, но они также **зависят от более простых метрик и не могут быть универсальными**.

Любая **метрика – это показатель, который зависит от языка и стиля программирования**. Поэтому **ни одна мера не может дать 100%-й гарантии в принятии решения** по программному средству, если опираться только на её значения.

6.2 МЕТРИКА Кокола

Метрика Кокола относится **к гибридным мерам. Метрики данного класса основываются на более простых метриках и представляют собой их взвешенную сумму**.

Метрика Кокола определяется:

$$HM = [(M + R_1 \cdot M(M_1) + \ldots + R_n \cdot M(M_n)] / (1 + R_1 + \ldots + R_n), \quad (6.1)$$

где M – **базовая метрика**;

M_n – иные интересующие меры;

R_n – подобранные (корректирующие) коэффициенты. Весовой коэффициент $R_Б$ базовой меры по Холстеду равен 1;

$M(M_n)$ – функции (корректирующие), отражающие зависимость базовой метрики от иных интересующих мер.

Функции $M(M_n)$ и коэффициенты R_n могут быть определены с помощью регрессионного анализа или непосредственного анализа задачи для конкретной программы.

Метрика Кокола использует три модели для мер: (**МакКейба**, **Холстеда** и ***SLOC***), где **в качестве базовой используется одна из мер Холстеда** (**одна из метрик по модели Холстеда**). Эти модели называют «**наилучшая**», «**случайная**» и «**линейная**».

SLOC – простая **метрика, определяющая количество строк** кода. Метрика была разработана для оценки трудозатрат.

Но так как одна и та же функциональность может быть разбита на несколько строк или записана в одну строку, метрика стала редко применимой с появлением языков, в которых в одну строку может быть записано больше одной команды. Поэтому **различают логические и физические строки кода**.

Логические строки кода – это **количество команд программы**. Этот **способ описания имеет** свои **недостатки, так как сильно зависит от используемого языка** программирования **и стиля программирования**.

Кроме *SLOC* могут быть использованы следующие количественные характеристики:

– **количество пустых строк**;

– **количество комментариев**;

– **процент комментариев (отношение числа строк, содержащих комментарии к общему количеству строк**, в процентах);

– **среднее число строк для функций** (классов, файлов);

– **среднее число строк, содержащих исходный код для функций** (классов, файлов);

– **среднее число строк для модулей**.

Используемые метрики в каждом варианте гибридных метрик **выбираются в зависимости от конкретной задачи. А для принятия решения в отдельном рассматриваемом случае, корректирующие коэффициенты выбираются в зависимости от значения базовой метрики и от степени влияния иных интересующих мер** на конечный результат метрики Кокола.

Ниже приведена мотивация расчётных факторов.

1. Знаменатель формулы (6.1), представляющий сумму весовых коэффициентов R_n для используемых мер, **указывает на необходимость нормализации найденных значений метрик**. Это также устраняет **суммирование размерных и безразмерных величин в формуле** (6.1), *например*, когда в качестве базовой меры по Холстеду выбран объём V программы с единицей измерения [**бит**].

2. В формуле (6.1) **весовой коэффициент $R_Б$ базовой меры по Холстеду** равен **1**. Для нормирования этой меры в ходе анализа программы (при выборе в качестве базовой меры «**объёма V программы**»), **в качестве нормы $V_{НОРМ}$** можно взять «**теоретический объём**» или «**потенциальный объём**» программы. **Нормирование** базовой меры «**объёма V программы**» **выполняется** известным способом:

$$M_{\text{Б НОРМ}} = V/V_{\text{НОРМ}}. \quad (6.2)$$

Полученное значение **нормированного значения $M_{\text{Б НОРМ}}$ базовой меры и используется при подсчёте значения метрики Кокола по формуле** (6.1) **в виде формулы** (6.3):

$$HM = [(M_{\text{Б НОРМ}} + R_1 \cdot M(M_1) + ... + R_n \cdot M(M_n)]/(1 + R_1 + ... + R_n). \quad (6.3)$$

3. Для нормирования меры по МакКейбу в ходе анализа программы, **в качестве нормы и получения нормированного значения меры $M_{1\text{Н}} = M(M_1)$**, можно взять **значения цикломатического числа Мак-Кейба из рекомендаций Института программной инженерии** (Пенсильвания, США. **См. лекцию 3**) **или из Руководства МакКоннелла** (книга «Code Complete», **Мак-Коннелл. См. лекцию 3**).

Учитывая, что **данная мера в составе гибридной метрики Кокола отвечает за сложность программного кода**, для этой меры можно принять **статистику по влиянию разветвлений в программе на качество её исходного кода** [10] **на основе метрики Fan Out** (**SFOUT;** развернуться веером). Эта мера для конкретной программы определяется **путём подсчёта количества** сущностей (*например*, функций, методов, классов, файлов и т. п.), **которые вызывает** или **от которых зависит** данная сущность.

Рассматривая **отдельную (автономную) программу, не связанную с другими модулями**, понятно, что эта мера определяется **путём подсчёта количества разветвлений** в программе.

При подсчёте **учитываются все прямые вызовы других** функций или методов. Если **одна и та же функция вызывается несколько раз, она засчитывается только один раз**. А на уровне файлов/модулей метрика **Fan Out** также **может включать зависимости от глобальных переменных, констант или структур данных, определенных в других модулях**.

То есть, **значение меры Fan Out для программы определяется**:

SFOUT = Количество процедур (модулей; блоков), вызываемых + Количество процедур (модулей; блоков), вызывающих. (6.4)

Рассматривая **отдельную программу, не связанную с другими модулями**, мера **Fan Out** для такой автономной программы **определяется путём** простого **подсчёта количества разветвлений** в программе. При этом **будут получены значения данной меры без необходимости определения среднего значения меры, учитывающего количество и связи внутренних и внешних** модулей.

По [10], статистика **влияния категорий метрики Fan Out на качество кода** отражена **в таблице 6.1**, в крайней правой колонке которой показан **для каждой категории возможный весовой коэффициент R_1 для метрики Кокола**.

Таблица 6.1 – **Влияние меры Fan Out на качество программы и** возможные весовые коэффициенты R_1 для метрики Кокола

Категория меры Fan Out	**Ухудшение качества кода, раз**	**Весовой коэффициент R_1 для метрики Кокола**
не более 3	1,11 и менее	1,11
более 3 и не более 6	1,25 и менее	1,25
более 6 и не более 10	1,43 и менее	1,43
более 10 и не более 20	2,00 и менее	2,00
более 20 и не более 26	2,50 и менее	2,50
более 26	более 2,50	более 2,50

Определение **поведения** корректирующей функции $M(M_1)$ для меры по МакКейбу **должно учитывать эмпирические (архивные) данные о взаимосвязи этой меры в предыдущих версиях программы, либо на предыдущих этапах жизненного цикла исследуемой программы**. Поскольку таких сведений не имеется, а **по числовой интерпретации значений меры Fan Out, сложность программы будет расти нелинейно.**

Используя метод наименьших квадратов для аппроксимации данных точками **(3; 1.11), (6:1.25), (10;1.43), (20;2), (26;2.5)** для **квадратичной функции** вида $y = ax^2 + bx + c$, получена **аппроксимация квадратичной функцией** вида: $y = 0.0011x^2 + 0.0276x + 1.032$. Так будет влиять поведение корректирующей функции $M(M_1)$ на меру Кокола.

Данное влияющее воздействие функции $M(M_1)$ уже заложено в значения весовых коэффициентов R_1, показанных **в таблице 6.1**. Поэтому, при подсчёте меры Кокола (**по формуле (6.5)**) нужно будет умножить выбранный весовой коэффициент **на рассчитанное значение меры $M_H(M_1)$.**

4. **Для нормирования меры *SLOC*** в ходе анализа программы, **для определения нормы можно исходить из следующих рассуждений**. **Количество строк кода** (Source Lines of Code («Исходные строки кода») – ***SLOC***) – метрика, используемая **для измерения объёма кода с помощью подсчёта количества строк** в тексте исходного кода. Этот показатель **обычно используется для прогноза трудозатрат на разработку программы** на конкретном языке программирования, либо **для оценки производительности труда** уже после того, как программа написана.

В метрике Кокола, где ***SLOC* использована как линейная мера**, обычно **используется подсчёт физических строк**. Однако, **результаты подсчёта сильно зависят от правил оформления и форматирования исходного кода**, чему **логические** строки кода **подвержены в гораздо меньшей степени**. И также появился ещё один аспект – **разница между программным кодом, написанным вручную, и сгенерированным автоматически**. **Современные средства разработки предоставляют возможность автоматически создавать большие объёмы кода** несколькими кликами мыши. То есть, **эта метрика целесообразна при оценке программ на языках, в которых одна строка кода реализует только одну команду**.

Но для метрики Кокола меру *SLOC* нужно рассматривать как метрику, отражающую степень понимания сложности и размера кода. Большое количество строк в одном файле **может указывать на его чрезмерную сложность и возможную необходимость рефакторинга для повышения читаемости и управляемости кодом. Разбиение на файлы меньшего размера даёт преимущество при слиянии кода. Также обеспечивается читабельность:** длинный файл очень сложно читать [7]. **Только**, примерно, **около 1000 строк кода отражает уровень сложности, который человек может удержать в голове одновременно**.

Обзорная оценка суждений специалистов–разработчиков программных систем [7–9] говорит, что **определять размер файла в строках кода нужно оценив назначение программы по следующим критериям**.

1. Направленность программы (разработка веб-приложений; мобильных приложений; СУБД; создание операционных систем, игр, инструментов искусственного интеллекта). **Направленность программы влияет на её сложность**.

2. Функциональная насыщенность программы. Чтобы компоненты программы оставались целостными, читабельными и простыми для понимания, **верхний предел строк в файле должен быть около 300 строк**, далее становится сложно разобраться. Наличие **300 строк в файле означает**, что в нём, вероятно, **есть несвязанные между собой функциональные элементы**, а это редко целесообразно.

Для **обеспечения сосредоточенности компонент, читабельности и понятности рекомендуется оптимальный размер примерно 150 строк** [8, 9]. Так оценивая **файлы, обеспечивающие научные вычисления, анализ данных и машинное обучение** – то для таких файлов на языке **Julia** указывается **число строк в диапазоне от 50 до 900**.

3. Файлы, обеспечивающие сложные вычисления – для таких файлов на языке **C++** указывается **число строк до 8500**. А **файлы, реализующие бизнес-логику и необходимую скорость развертывания программного продукта, имеют большое количество бизнес-правил**. Здесь возможно **1000 и более строк кода** [8, 9]. Простота понимания той или иной функции определяется сложностью бизнес-процессов.

Поэтому, исследуя программу с помощью метрики Кокола, для меры ***SLOC*** **нужно выбирать значение нормы $M_{\text{Н}}$, опираясь на примерную оценку**, данную **в таблице 6.2**.

Таблица 6.2 – **Примерные значения нормы для меры *SLOC*** в зависимости от сложности, функциональной и логической насыщенности программ

Характеристика программ	**Диапазон значений для нормы M_H, строк**	**Рекомендуется M_H, строк (среднее геометрическое)**	**Весовой коэффициент R_2 для метрики Кокола***
Простые (со связанными между собой функциональными элементами)	20–150	55	0,00055
Повышенной сложности (в том числе с наличием несвязанных между собой функциональных элементов)	150–1000	390	0,00389
Высокой сложности (сложные вычисления; реализация бизнес-логики и бизнес-процессов)	1000–8500	2900	0,02858

***весовой коэффициент R_2 для метрики Кокола определён при вероятности наличия остаточной ошибки в строке кода равной 0,00001 = 10^{-5}, то есть 0,001 %.**

Предположим, что исследуемая **программа прошла всестороннее тестирование и имеет вероятность наличия остаточной ошибки в строке кода равна 0,001 % = 0,00001**. Тогда **вероятность отсутствия ошибки в одной строке равна 1 – 0,00001 = 0,99999**. И **вероятность отсутствия ошибки во всей программе** будет **$0{,}99999^{SLOC}$**. А **вероятность присутствия хотя бы одной ошибки в программе** будет равна **$1 - 0{,}99999^{SLOC}$**.

Отсюда, как **вероятность наличия остаточной ошибки в строке кода,** определено **значение весового коэффициента R_2 для метрики Кокола, внесённое в таблицу 6.2.**

Если для исследуемой программы, по мнению аналитика, **число физических строк кода мало без учёта пустых строк**, то можно учесть **и пустые строки (если их не более 25 % от общего числа строк**).

Поведение корректирующей функции $M(M_2)$ для меры ***SLOC*, по определению автора данной модели объединения метрик Петера Кокола** (Peter Kokol), **является линейной** [6]. Для функции $M(M_2)$ **можно принять** поведение функции как **линейное.**

Предполагается, что эта линейная функция $M(M_2)$ имеет вид $y = ax^b$. **Для определения параметров a** и **b** использовано логарифмирование по натуральному основанию, что даст линейное уравнение. **Для заданных точек из таблицы 6.2** вычисляются натуральные логарифмы. Используя две точки,

строится и решается система линейных уравнений (из двух уравнений), и получена функция: $y = 0{,}00001x^{0{,}998989}$.

То есть, поведение функции $M(M_2)$ имеет вид линейной функции $y = 0{,}00001x^{0{,}998989}$. Данное влияющее воздействие функции $M(M_2)$ уже заложено в значения весовых коэффициентов R_2, показанных **в таблице 6.2**. Поэтому, при подсчёте меры Кокола (**по формуле (6.5)**) нужно будет умножить выбранный весовой коэффициент **на рассчитанное значение меры** $M_H(M_2)$**.**

5. Итоговая формула для расчёта меры Кокола в данной работе будет иметь вид (формула (6.5):

$$HM = [(M_{Б\ НОРМ} + R_1 \cdot [M_H(M_1)] + R_2 \cdot [M_H(M_2)] \,/\, (1 + R_1 + R_2). \quad (6.5)$$

6.3 ЗАДАНИЕ

1. Выбрать программу, в которой ОБЯЗАТЕЛЬНО должны быть ВЕТВЛЕНИЯ, или ЦИКЛЫ (или то и другое). Представить графически схему алгоритма программы [1]. **Установочный материал см. в приложении А.**

2. Определить, по модели Холстеда, количественные параметры исследуемой программы, связанные с её размером. Из них выбрать меру, принимаемую за базовую («наилучшая») в метрике Кокола . Провести нормализацию соответствующего параметра исследуемой программы.

В качестве мер по модели Холстеда рекомендуется определить следующие:

1) число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);

2) число уникальных операндов программы (словарь операндов);

3) общее число операторов в программе;

4) общее число операндов в программе;

5) алфавит (**словарь**) программы;

6) экспериментальную длину программы;

7) теоретическую длину программы;

8) объём программы;

9) потенциальный объём программы.

Пример **расчёта параметров исследуемой программы** и **нормализации выбранного параметра** дан **в приложении Б**.

3. Представить поток управления в программе в виде графа. Рассчитать меру МакКейба (цикломатическую сложность графа потока управления), являющуюся **мерой M_1 («случайная») в метрике Кокола. Провести нормализацию значения меры M_1. Определить весовой коэффициент R_1 для метрики Кокола**.

Пример **построения «графа потока управления» и расчёта** приведён **в приложении В**.

4. Рассчитать значение метрики *SLOC*, являющуюся **мерой M_2 («линейная») в метрике Кокола. Провести нормализацию значения меры M_2. Определить весовой коэффициент R_2 для метрики Кокола**.

Пример **расчёта меры *SLOC*** приведён **в приложении Г**.

5. Рассчитать значение гибридной метрики Кокола *HM* для выбранной программы. Программа индивидуальна для каждого студента.

Пример **расчёта меры *HM*** приведён **в приложении Д**.

6. Дать собственную смысловую оценку полученного значения и составляющих меры *HM* гибридной метрики Кокола.

Пример **оценки меры *HM*** приведён **в приложении Е**.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 19.701-90. ЕСПД. Схемы алгоритмов, программ, данных и систем. – М.: Изд-во стандартов, 1990. – 8 с.

2. Звездин С.В. Проблемы измерения качества программного кода // Вестник ЮУрГУ. – Сер. «Компьютерные технологии, управление, радиоэлектроника». – Вып. 11. – № 2. – 2010. – С. 62–66.

3. Качество программного обеспечения (Software Quality). – URL: https://analytics.infozone.pro/software-quality/ (дата обращения 12.06.2019).

4. Петрухин В.А., Лаврищева Е.М. Методы и средства инженерии программного обеспечения. – URL: http://www.intuit.ru/studies/courses/2190/237/lecture/6136?page=5 (дата обращения 02.04.2018).

5. Черников Б.В., Поклонов Б.Е. Оценка качества программного обеспечения. Практикум. – М., ИД «Форум»–ИНФРА–М. – 2012. – 400 с.

6. Орлов С.А. Технологии разработки программного обеспечения. – С-Пб.: Питер, 2002. – 322 с.

7. Do we have standard for LOC per file?. – URL: https://users.rust-lang.org/t/do-we-have-standard-for-loc-per-file/63509/3 (дата обращения 03.08.2023).

8. How many lines of code is too much?. – URL: https://www.linkedin.com/posts/jomkit-jujaroen_softwareengineering-codequality-refactoring-activity-7381737063198543872-gt0Z (дата обращения 03.08.2023).

9. What is the optimal code line length per file. – URL: https://forum.cursor.com/t/what-is-the-optimal-code-line-length-per-file/120458 (дата обращения 28.08.2025).

10. Enas Alikhashashneh, Rajeev Raje, James Hill. Using Software Engineering Metrics to Evaluate the Quality of Static Code Analysis Tools. – URL: https://ieeexplore.ieee.org/document/8367641 (дата обращения 14.06.2021).

11.

ПРИЛОЖЕНИЕ А

Графическое представление схемы алгоритма программы

1) представить рисунком схему алгоритма программы, выполненной

на знакомом языке программирования (одном языке).

Размер листинга не более одной–двух страниц формата А4, индивидуален для каждого студента.

Пример программы алгоритма схемы приведён на рисунке А.1, [1].

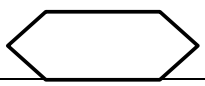
Схема алгоритма программы должна отражать последовательность операций в программе, а не показывать схему работы какой-то системы.

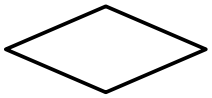
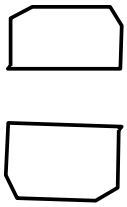
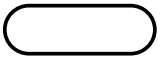
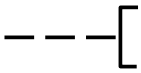
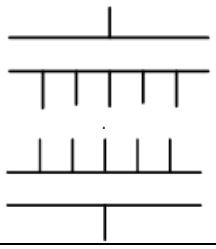
Не использовать синтаксис языка программирования, и при необходимости руководствоваться п. 4.1.4 ГОСТ [1] («Если объем текста, помещаемого внутри символа, превышает его размеры, следует использовать символ комментария. Если использование символов комментария может запутать или разрушить ход схемы, текст следует помещать на отдельном листе и давать перекрестную ссылку на символ»).

Также, п. 4.1.5 ГОСТ [1]: («В схемах может использоваться идентификатор символов. Это связанный с данным символом идентификатор, который определяет символ для использования в справочных целях в других элементах документации (например, в листинге программы). Идентификатор символа должен располагаться слева над символом»).

Условные графические обозначения (УГО) и правила выполнения по ГОСТ – краткая справка в таблице А.1.

Таблица А.1 – УГО в схемах алгоритмов, программ, данных и систем

УГО	Наименование	Назначение
	**Данные**	Определяет ввод или вывод на внешнее устройство или носитель данных.
	**Процесс**	Отражает обработку данных (выполнение операции, группы операций).
	**Типовой процесс**	Отображает предопределенный процесс, состоящий из одной или нескольких операций программы, которые определены в другом ме-
	**Подготовка**	Отображает модификацию параметра для управления циклом со счет-

	**Решение**	Описывает проверку условия и выполняет переключение по одному из условий. Имеет один вход и два или более альтернативных выходов, один из которых активизируется после вычисления условия.
	**Граница цикла**	Состоит из двух частей: начала и конца цикла. Обе части имеют один и тот же идентификатор. Изменение значения идентификатора, условия для выполнения или завершения помещаются внутри символов в начале или в конце цикла.
	**Соединитель**	Используется для обрыва линии и продолжения ее в другом месте.
	**Знак завершения**	Определяет начало и конец структурной схемы алгоритма.
	**Комментарий**	Используется для добавления пояснительных записей.
	Основная линия	Отражает последовательность выполнения действий в алгоритме.
	начало **Параллельные действия** конец	Применяется в случае одновременного выполнения операций, отображаемых несколькими символами.

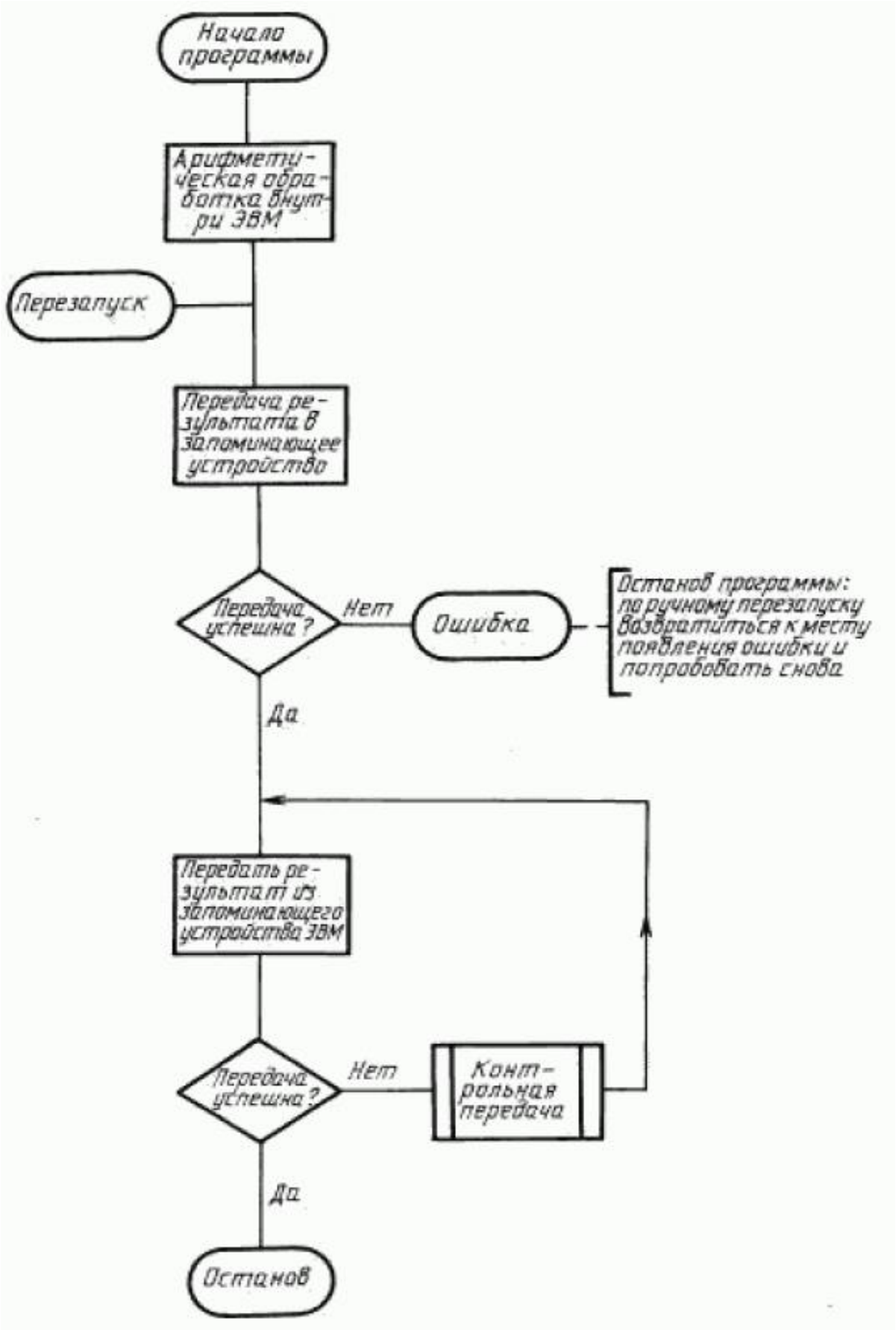


Рисунок А.1 – Пример схемы программы (алгоритма)

ПРИЛОЖЕНИЕ Б

Расчёт параметров программы по модели Холстеда и нормализация выбранного параметра

Проанализировав листинг исследуемой (**некой**) программы **по модели Холстеда**, и опираясь на синтаксис языка, **получили результаты, показанные в таблицах Б.1, Б.2** и **Б.3.**

Листинг приводить в приложении к основной части отчёта по ПЗ6, а не в теле его основной части.

Результаты подсчёта числа типов операторов и операндов для **некой** рассматриваемой программы, и их общего количества **представлены в таблицах Б.1 и 2.2**.

Таблица Б.1 – Таблица операторов программы

Оператор	Уникальный n_1	N_1
;	1	20
{}	2	12
()	3	46
=	4	10
+	5	1
++	6	1
[]	7	1
for (int i = 0; i < 2 * N; i++)	8	1
new	9	1
Console.WriteLine	10	6
Convert.ToInt32	11	3
Console.ReadLine	12	3
Console.Write	13	9
Console.ReadKey	14	1
<	15	1
void	16	1
static	17	1
try	18	1
catch	19	1
*	22	2
+=	23	1
	23	123
	$n_1 = 23$	$N_1 = 123$

Таблица Б.2 – Таблица операндов программы

Операнд	**Уникальный n_2**	**N_2**
string	1	2
int	2	5
var	3	1
a	4	3
c	5	3
N	6	4
x	7	4
x0	8	3
arr	9	3
i	10	4
e	11	2
Exception	12	1
numb	13	3
"Введите параметры ЛКГ"	14	1
"a = "	15	1
"c = "	16	1
"N = "	17	1
"Идет генерация чисел..."	18	1
"Numbers.txt"	19	1
""	20	1
";"	21	1
"Генерация завершена, результат находится в файле Numbers.txt в каталоге программы"	22	1
	22	47
	$n_2 = 22$	$N_2 = 47$

По итогам подсчёта числа операторов и операндов, полученные результаты сведены в таблицу Б.3.

Таблица Б.3 – Исходные данные для определения метрик Холстеда

Название данных	**Частота данных**
Число уникальных операторов (n_1)	23
Число уникальных операндов (n_2)	22
Общее число операторов (N_1)	123
Общее число операндов (N_2)	47

Словарь программы (n)	45

Мера по модели Холстеда является базовой в метрике Кокола и должна свидетельствовать об объёмных характеристиках программы. Значение выбранной меры должно быть способно информировать разработчиков о том, что какой-либо блок или класс содержит слишком большое или слишком маленькое количество методов и необходимо пересмотреть данный участок кода.

Поэтому, в ходе смысловой оценки исследуемой программы можно заключить: из мер по модели Холстеда можно остановиться на той, которая характеризуют программу как наиболее значимый параметр характеристики «объём кода».

Учитывая указанные выше значения исходных данных для расчёта по модели Холстеда, таблица Б.3, находим следующие оценки.

Словарь программы, определяется по формуле:

$$n = n_1 + n_2 = 23 + 22 = 45. \quad (Б.1)$$

Длина исследуемой программы, определяется по формуле:

$$N = N_1 + N_2 = 123 + 47 = 170. \quad (Б.2)$$

Объём исследуемой программы определяется как:

$$V = N log_2(n) = 170 log_2(45) = 933{,}62 \text{ бит}. \quad (Б.3)$$

В то же время, теоретический объём исследуемой программы определяется как:

$$V_T = n_1 log_2(n_1) + n_2 log_2(n_2) = 23 log_2(23) + 22 log_2(22) = 202{,}15 \text{ бит}. \quad (Б.4)$$

Потенциальный объём программы, определяется как:

$$V^* = n log_2(n) = 45 log_2(45) = 247{,}13 \text{ бит}. \quad (Б.5)$$

Значение меры «**объём программы**» $\boldsymbol{V^*}$ **в записи на потенциальном языке**, по Холстеду, должно иметь минимально возможное значение, так как слова (операнды и операторы) в идеальной программе не повторяются, и на потенциальном языке «длина программы» совпадает с «объёмом программы».

Но в выполненном расчёте значение меры $\boldsymbol{V_T}$ «**теоретический объём**» оказалось меньше, чем значение меры «потенциальный объём» программы, что можно объяснить значительной разностью уровней современных языков и технологий программирования по отношению ко времени создания метрик Холстеда.

Поэтому в качестве нормы $\boldsymbol{V_{НОРМ}}$ параметра, характеризующего размер программы, выбрана мера «**теоретический объём**», имеющая значение **202,15 бит**. С её помощью проведём нормирование параметра $\boldsymbol{V}$ «**объём** иссле-

дуемой **программы**».

Нормирование базовой меры – параметра ***V*** «**объём программы**» – для метрики Кокола «**объём *V* программы**» **выполним** по формуле:

$$M_{Б\ НОРМ} = V/V_{НОРМ} = 933{,}62 / 202{,}15 = 4{,}62. \quad (Б.6)$$

Полученное значение **нормированного значения $M_{Б\ НОРМ}$ базовой меры и используем при подсчёте значения метрики Кокола по формуле** (6.4).

ПРИЛОЖЕНИЕ В

Построение графа потока управления и расчёт меры M_1 для метрики Кокола

Ниже приведён **граф потока управления (рисунок В.1) НЕКОТОРОЙ** программы, **для которого выполнен расчёт** меры M_1 **для метрики Кокола**.

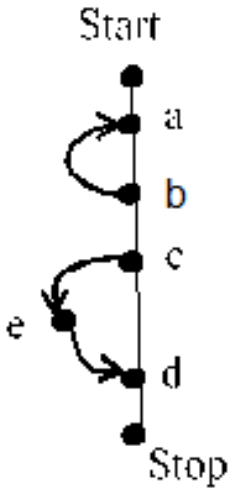


Рисунок В.1 – **Граф потока управления некоторой программы**

Для вычисления цикломатической сложности графа потока управления используем формулу (В.1). Характеристики графа приведены **в таблице В.1**.

Таблица В.1 – **Характеристики графа потока управления**

Характеристика	Значение
Количество дуг	8
Количество вершин	7
Компонента связности	1

Подставив все значения в формулу (В.1), цикломатическая сложность графа потока управления будет равна трём: $Z(G) = 3$:

$$Z(G) = e - v + 2p = 8 - 7 + 2 \cdot 1 = 3, \quad \text{(В.1)}$$

где e – количество дуг графа;

v – количество вершин графа;

p – количество компонент связности.

Для нормирования этой **меры по МакКейбу (**получение нормированного значения для метрики Кокола**), в качестве нормы** $M_{1\,\text{НОРМ}}$ взято **значение цикломатического числа МакКейба из «Руководства МакКоннелла** (книга «Code Complete»», **лекция 3**), **равное 5 (**$M_{1\,\text{НОРМ}} = 5$**)**, которое **трактует исследуемую программу как процедуру, не перегруженную излишней сложностью**.

Нормирование «случайной» меры – параметра M_1 для метрики Кокола, **выполним** по формуле:

$$M_{\text{Н}}(M_1) = Z(G) / M_{1\,\text{НОРМ}} = 3 / 5 = 0{,}6. \quad \text{(В.2)}$$

Данная мера в составе гибридной метрики Кокола отвечает за сложность программного кода, и для этой меры, так как **на рисунке В.1** количество разветвлений в программе – **два**, то **можно принять, используя таблицу 6.1, значение нормированного весового коэффициента** R_1 **(для формулы (6.5), равным 1,11:** $R_1 = 1{,}11$.

Поведение корректирующей функции $M(M_1)$ для меры по МакКейбу в **формуле (6.5) указывает, что сложность программы будет расти нелинейно и согласно квадратичной функции** вида: $y = 0.0011x^2 + 0.0276x + 1.032$.

Полученные величины **нормированного значения базовой меры** $M_{\text{Н}}(M_1) = 0{,}6$ **и весового коэффициента** $R_1 = 1{,}11$ **используем при подсчёте значения метрики Кокола по формуле** (6.5).

ПРИЛОЖЕНИЕ Г

Расчёт меры M_2 для метрики Кокола

Количество строк кода (Source Lines of Code – ***SLOC***) – это метрика, используемая для измерения объёма кода с помощью подсчёта количества строк в тексте исходного кода. В метрике Кокола мера ***SLOC*** **использована как линейная мера, в которой обычно используется подсчёт числа физических строк**. **Физические строки** – это **непустые строки листинга**. Пустые строки учитываются в том случае, если их количество не превышает 25 % от общего числа строк.

Для **НЕКОЙ** исследуемой программы **число физических строк кода оказалось равным 46 без учёта пустых строк:** ***SLOC*** **= 46**. Так как количество строк кода мало, и учитывая то, что **данная мера в метрике Кокола интерпретирует сложность программы в зависимости от числа строк кода**, можно считать её влияние на итоговое значение гибридной метрики исследуемой программы крайне незначительным. **На основе таблицы 6.2 считаем**:

1) программа простая, со связанными между собой функциональными элементами, и **при числе строк кода** ***SLOC*** **= 46 выбираем значение номы** $M_{2\,\text{НОРМ}}$ равным **55 строкам**: **(**$M_{2\,\text{НОРМ}} = 55$**)**.

Нормирование «линейной» меры – параметра M_2 для метрики Кокола, **выполним** по формуле:

$$M_{\text{Н}}(M_2) = SLOC \,/\, M_{2\,\text{НОРМ}} = 46 \,/\, 55 = 0{,}84. \qquad (\text{Г}.1)$$

2) Используя таблицу 6.2, значение нормированного весового коэффициента R_2 (**для формулы (6.5)** выбираем **равным 0,00055:** $R_2 = 0{,}00055$.

Поведение корректирующей функции $M(M_2)$ для меры ***SLOC*** в **формуле (6.5) указывает, что сложность программы будет расти линейно и согласно линейной функции** вида**:** $y = 0{,}00001x^{0{,}998989}$. Это **график прямо пропорциональной зависимости** (степень аргумента очень близка к единице (**0,998989**). **Коэффициент пропорциональности** графика функции также **крайне мал** (**0,00001**), график **почти совпадает с осью абсцисс**.

То есть, **при вероятности наличия остаточной ошибки в строке кода равной 0,00001 =** 10^{-5}**, или 0,001 %**, для исследуемой программы **влияние корректирующей метрики *SLOC* на меру Кокола исчезающе мало.**

Полученные величины **нормированного значения «линейной»**

меры $M_{Н}(M_2) = 0,84$ и весового коэффициента $R_2 = 0,00055$ используем при подсчёте значения метрики Кокола по формуле (6.5).

ПРИЛОЖЕНИЕ Д

Расчёт значения *НМ* гибридной метрики Кокола

Значение гибридной меры Кокола *НМ* для исследуемой программы, с учётом мотивированных выше корректирующих весовых коэффициентов и корректирующих функций для составляющих в формуле (6.5), определится:

$$НМ = [(M_{Б\ НОРМ} + R_1 \cdot [M_{Н}(M_1)] + R_2 \cdot [M_{Н}(M_2)] \,/\, (1 + R_1 + R_2) =$$

$$= [(1 \cdot 4,62 + 1,11 \cdot [0,6] + 0,00055 \cdot [0,84]] \,/\, (1 + 1,11 + 0,00055) = 2,50478. \quad (6.6)$$

Таким образом, получено **значение объединяющей меры** по метрике Кокола, равное **2,50478**. Это значение отражает взвешенную сумму – **единственное числовое значение для набора метрик с учётом взвешенных коэффициентов**.

Исходя из метрик, использованных в мере Кокола, видно, что они позволяют аналитику оценить как объём работ и их трудоёмкость (метрика ***SLOC***), так и сложность программного продукта за счёт суммы мер **объёма** программы, **цикломатического числа МакКейба** и ***SLOC*** .

Наблюдение за значением гибридной метрики на временном интервале реализации проекта позволяет контролировать качество разработки и судить о допущенных несовершенствах и ошибках при резких изменениях значения гибридной метрики.

Решение, какие именно метрики будут применяться в составе гибридной меры, зависит от конкретного проекта, в котором могут быть использованы те или иные метрики.

ПРИЛОЖЕНИЕ Е

Оценка значения и составляющих меры *НМ* гибридной метрики Кокола

По метрике Холстеда. В рассчитанном значении метрики Кокола (2,50478) базовая мера по Холстеду, отражающая объёмную характеристику кода, составляет взвешенную величину, равную 4,62 / (1 + R_1 + R_2) = 4,62 / (1 + 1,11 + 0,00055) = 2,18900, т. е. это составляет 87,39 % от значения меры *НМ* = 2,50478.

Отсюда следует, что уровень качества исследуемой программы определяется её объёмом.

По Холстеду, размер метрики «объём программы» для одного файла рекомендуется от 100 до 8000 бит. Если значение меры превышает верхнюю границу указанного диапазона, то рекомендуется разбить исследуемый программный модуль на несколько составляющих.

В исследуемой программе объём составляет 933,62 бит и укладывается в рекомендуемый диапазон.

По метрике Мак-Кейба. Цикломатическое число МакКейба для исследуемой программы (равно 3) меньше критерия $\boldsymbol{Z(G) = 5}$, а значит программа не перегружена излишней сложностью. Программу не нужно разбивать на составные части с меньшим значением цикломатического числа.

В рассчитанном значении метрики Кокола (2,50478), мера по МакКейбу, отражающая сложность кода, составляет взвешенную величину: 0,66 / (1 + R_1 + + R_2) = 0,66 / (1 + 1,11 + 0,00055) = 0,31271, т. е. это составляет 12,48 % от значения меры *НМ* = 2,50478.

Отсюда следует, что уровень качества исследуемой программы определяется сложностью потока управления в программе, но в меньшей степени, чем объёмом программы (меньше в 87,39/12,48 = 7,0 раз).

По метрике SLOC. Для исследуемой программы число физических строк равно 46. Из-за небольшого количества строк кода можно считать, что трудозатраты на написание программы малы, и вероятность необнаруженной ошибки в коде программы также крайне мала.

В рассчитанном значении метрики Кокола (2,50478), мера по SLOC, отражающая объёмность (громоздкость) кода, составляет взвешенную величину: 0,000462 / (1 + R_1 + R_2) = 0,000462 / (1 + 1,11 + 0,00055) = 0,00022, т. е. это составляет 0,009 % от значения меры *НМ* = 2,50478.

.

То есть, это означает, что уровень качества исследуемой программы не

определяется количеством её физических строк. Отличие от меры Холстеда в 87,39/0,009 = 9710 раз, а отличие от меры МакКейба в 12,48 /0,009 = 1387 раз.

По полученному значению метрики Кокола. Значение этой метрики (2,50478), отражающее взвешенную сумму базовой меры с добавленными более простыми метриками, и приведенная выше оценка вклада простых мер (суммарно (12,48 + 0,009 = 12,489) %, говорит разработчику, что качество исследуемой программы определяется в большой степени её объёмом, в меньшей степени (в 7 раз) наличием циклов в программе, и фактически не зависит от наличествующего количества строк кода.

Указанное позволяет оценить и объём работ на создание программы, и их трудоёмкость, а также сложность исследованного программного продукта.