УДК 004.052.4

С.А. ЯРЕМЧУК

Одесский национальный политехнический университет, Украина

МЕТОД ОЦЕНКИ КОЛИЧЕСТВА ПРОГРАММНЫХ ДЕФЕКТОВ С ИСПОЛЬЗОВАНИЕМ МЕТРИК СЛОЖНОСТИ

В работе выполнен критический анализ наиболее часто используемых метрик программного обеспечения. Определена степень возможности использования метрик размера и сложности для оценки количества программных дефектов. Для более точной оценки сложности метрики Мак-Кейба и Чепина дополнены ранее не учтенными аспектами программной сложности. Выявлено отсутствие метрики сложности вычислений. Произведены подсчеты метрик для семи программных проектов. Предложен метод оценки количества программных дефектов с использованием метрик сложности. Для проверки метода выполнены расчеты оценки дефектов, определена точность оценок метода, указаны его преимущества.

***Ключевые слова**: сложность программ, дефекты программного обеспечения, метрики размера, метрики сложности, метод оценки количества программных дефектов.*

Введение

Характерной особенностью современного этапа развития является возрастающая зависимость человечества от надежности работы программного обеспечения (ПО) в составе разнообразных информационных систем (ИС).

В работе [1] показано, что только 5% дефектов ИС связаны с аппаратурой, остальные 95% - ошибки разработки ПО. Расчеты динамики роста объема ПО в тыс.строк исходного кода говорят о том, что за десять лет (1980-1990г.) объем вырос с 10 000 до 50 000 строк кода, т.е. в 5 раз. За шесть лет (1997-2003г.) объем вырос с 75 000 до 600 000 строк кода, т.е. в 8 раз. Т.о., темп роста объемов ПО в 2000 г. по сравнению с 1990 г. увеличился в 2,7 раза [2].

Э. Дейкстра [3] обращал внимание на то, что компьютерные технологии заставляют человеческий разум охватывать диапазон от отдельных битов до сотен мегабайт информации, что соответствует разнице в девять порядков. За прошедшее время сложность ПО намного выросла, и сегодня отношение Дейкстры вполне может характеризоваться 15 порядками [4]. Растущие объемы и сложность являются основными причинами дефектов в ПО.

Размеры и сложность ПО отражены многочисленными метриками, разработанными многими исследователями.

В [5] метрика ПО определяется как мера некоторого свойства, имеющая численное значение.

В [4] отмечено, «что сложность управляющей логики программы коррелирует с частыми ошибками и низкой надежностью. Обнаружены и другие факторы, например, количество переменных программы».

В работе [6] авторами указано на неоднозначность соответствий между значениями метрик и свойствами надежности.

Использование метрик для оценки дефектов ПО является вполне логичным, но малоизученным вопросом. Поэтому исследования метрик, и на этой основе разработка метода оценки программных ошибок является актуальной научно-технической задачей.

Постановка задачи

Цель работы – разработать метод оценки программных дефектов с использованием метрик ПО.

Для достижения поставленной цели необходимо выполнение следующих задач:

1. Выполнить анализ размерных метрик и определить возможность их использования для оценки программных дефектов;

2. Исследовать метрики Холстеда и точность оценки дефектов;

3. Исследовать другие наиболее известные метрики сложности, и возможность их использования для оценки дефектов;

4. На основе проведенных исследований предложить метод оценки программных дефектов. Выполнить расчеты проверки метода и определить точность оценок.

Результаты исследований

Метрики ориентированы на статический анализ исходного текста программы. Поскольку количество дефектов зависит от размеров и сложности ПО, будут рассмотрены две основные группы метрик: метрики размера и метрики сложности. Метрики первой группы определяют количественные характеристики размеров ПО, и отличаются относительной простотой. К наиболее известным метрикам данной группы относятся количество строк программного кода и метрики Холстеда.

1. Размерные метрики

LOC-оценка (Lines Of Code) является наиболее простой и распространенной метрикой размера ПО. Известно, что количество строк исходного кода существенно зависит от языка программирования, количества пустых строк и комментариев, возможностей современных сред разработки автоматически генерировать код, и других факторов.

Анализируя статистические данные электронного ресурса [7] в виде LOC-оценки и количества дефектов более **120** программных проектов разных разработчиков, был установлен **коэффициент корреляции 0,677,** отражающий далеко не тесную связь между размерами проектов в LOC и количеством дефектов. Вид зависимости KLOC (тыс. строк) и дефектов представлен на рисунке 1.

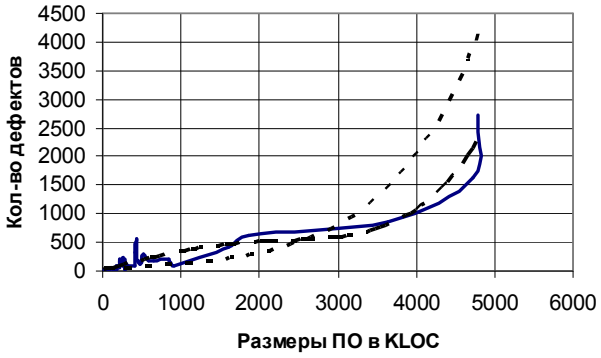

Рис. 1. Зависимость KLOC и дефектов ПО

Становится очевидной нелинейность зависимости между размерами ПО и количеством дефектов. Пунктирными линиями на рисунке обозначен экспоненциальный тренд, и тренд полинома 4-ой степени, наиболее близкий к экспериментальной кривой.

Также обращает на себя внимание тот факт, что при размере проекта, близком к 5 млн. строк, начинается резкий рост количества дефектов. Это можно объяснить значительно возрастающей сложностью в масштабных проектах.

Учитывая нелинейность зависимости, проблемным представляется применение некоторых априорных моделей надежности ПО.

Модель Акиямы [8] является одной из первых моделей для подсчета количества дефектов в ПО по количеству строк:

$$B = 4,86 + 0,018L, \qquad (1)$$

где B - общее количество дефектов в ПО, L – количество строк в программном коде.

Модель Гафнии [9] имеет следующий вид:

$$B = 4,2 + 0,0015(L)^{4/3}. \qquad (2)$$

Результаты расчетов по данным моделям на основе данных ресурса [7] представлены в табл. 1.

Таблица 1

Результаты расчетов
по моделям Акиямы и Гафнии.

К-во строк кода	Факт. к-во дефектов	Дефекты по Акияме	Дефекты по Гафнии
57 351	27	1037	3322
81 569	23	1473	5310
82 229	37	1485	5368
85 921	59	1551	5691
100 068	36	1806	6973
124 183	66	2240	9298

Несложные вычисления показывают расхождение фактических и расчетных данных в десятки раз для модели Акиямы и в сотни раз для модели Гафнии. Такое большое расхождение объясняется длительным периодом времени, прошедшем со времени разработки моделей. За прошедшие более 30 лет методы, средства и технологии программной инженерии многократно развились, и уровень ошибок в ПО значительно снизился. Поэтому данные модели потеряли свою актуальность, и их оценка дефектов является недостоверной. Поскольку LOC-оценка оценивает ПО линейно, без учета сложности, связь между LOC и количеством дефектов не тесная, зависимость не однозначная, поэтому LOC-оценка не является универсальным показателем, приемлемым для оценки программных дефектов.

2. Метрики Холстеда

Наиболее известными и ставшие классическими, внесшими большой вклад в развитие теории и практики разработки ПО, являются метрики Холстеда, предложенные автором еще в 1977 году [10]. Из предложенных автором метрик будут использованы те, которые непосредственно участвуют в оценке дефектов. Это **словарь** программы

$$n = n_1 + n_2 , \quad (3)$$

где n_1 - число простых уникальных операторов, n_2 - число простых уникальных операндов. Это **длина** программы

$$N = N_1 + N_2 , \quad (4)$$

где N_1 - общее число всех операторов, N_2 - общее число всех операндов. Это **объем** программы

$$V = N * \log_2 n . \quad (5)$$

Это оценка числа **дефектов** в программе

$$B = V / 3000 . \quad (6)$$

Для экспериментального определения значений метрик из выражений (3) и (4) были использованы исходные тексты 7-ми небольших учетно-аналитических программных проектов некритического применения, разработанных с участием автора, с известным фактическим числом дефектов, и инструментальные средства среды разработки.

Необходимо заметить, что метод подсчета значений n_1, n_2, N_1, N_2, предложенный Холстедом и основанный на классификации операторов и операндов, имеет высокую трудоемкость.

Особенную трудность вызывают подсчеты значений N_1, N_2 в связи с большим их количеством.

Для подсчета метрик в масштабных проектах необходимы статические анализаторы кода, гибко настраиваемые на особенности языка программирования и перечень уникальных операндов программы.

Данные метрики подсчитывались согласно выражению:

$$S_{sum} = \sum_{i=1}^{k} S_{mod} , \quad (7)$$

где i -количество модулей в ПО.

Результаты подсчетов и вычислений приведены в табл. 2.

Таблица 2

Экспериментальные результаты расчетов метрик Холстеда.

Метрика	Проект1	Проект2	Проект3	Проект4	Проект5	Проект6	Проект7	Коррел. метрики и факт. дефектов
N_1	605	828	11 300	8 208	14 563	21 670	25 013	0,9912
n_1	80	96	120	115	123	132	148	0,9593
N_2	641	741	7853	5315	9354	13466	17016	0,9857
n_2	180	92	1388	513	1628	1956	2128	0,9399
$n = n_1 + n_2$	260	188	1508	628	1751	2088	2276	0,9424
$N = N_1 + N_2$	1 246	1 569	19 153	13 523	23 917	35 136	42 029	0,9897
$V = N * \log_2 n$	9 996	11 853	202 225	125 691	257 681	387 477	468 719	0,9849
Расчетное число дефектов	3,33	3,95	67,41	41,90	85,89	129,16	156,24	
Фактическое число дефектов	4	6	36	39	57	76	89	
% отклонений	-17%	-34%	87%	7%	51%	70%	76%	

Несмотря на высокий коэффициент корреляции значений метрик и фактического числа дефектов, наибольшее отклонение составило 87%, среднее - 49%. Для маленьких программ модель занижает количество дефектов, для более крупных - завышает это количество.

Специфика исследуемых программных проектов заключается в большой степени повторного использования безошибочного кода, что в настоящий момент является обычной и распространенной практикой. Однако этот аспект был неизвестен в 1977 году, и данными метриками не учитывается. Проверка метрик выполнялась разными исследователями и давала положительные результаты для программ, написанных более 35 лет назад. В то время еще не существовали развитые средства разработки, включающие проверку синтаксиса при написании кода, высокий уровень предупреждений и сообще-

ний компиляторов, анализаторы кода и другие средства, препятствующие внесению ошибок в код во время разработки.

Поскольку все операторы по Холстеду имеют одинаковый вес, программа со сложной графовой структурой и структурно линейная программа, могут иметь равные объемы и количество ошибок. А это неверно.

С учетом высокой трудоемкости и значительных отклонений использование данных метрик в настоящее время является мало эффективным и мало достоверным средством оценки дефектов.

3. Метрики сложности

Известно, что структура любой программы состоит из следующих базовых программных компонентов: структур данных; структур управляющей логики, вычислительных структур, а также структур межмодульных связей. По этой причине из множества метрик, детально описанных авторами в работе [11], были выбраны метрики, количественно выражающие сложность этих основных программных структур**.** Немаловажное значение при выборе метрик имела простота автоматизированного подсчета значений метрик, а также их применимость для структурных и объектно-ориентированных программных проектов.

Метрика цикломатической сложности Мак-Кейба является одним из наиболее распространенных показателей оценки сложности ПО. Данный показатель вычисляется на основе графа управляющей логики программы, и является показателем сложности структур управления. Для автоматизированного вычисления показателя применяется упрощенный подход, в соответствии с которым построение графа не осуществляется, а вычисление показателя производится по методике, предложенной автором метрики [12]: 1. Начните считать с 1 на линейном участке кода; 2. Добавляйте 1 для каждого из следующих ключевых слов: if, while, for, and, or; 3. Добавляйте 1 для каждого варианта в операторе case.

Данная методика учитывает все виды операторов циклов и условий, а также сложность их логических предикатов. Однако для полной оценки сложности в методику необходимо добавить увеличение сложности за счет вложенности операторов, за счет увеличением длины операторов циклов и условий. Вложенность обсуждается в компьютерной литературе на протяжении 25 лет, и является одной из причин внесения в код ошибок. Длина цикла или условия, превышающая зону видимости, также служит частой причиной ошибок [4]. С учетом предложенных дополнений и принятого двукратного увеличения сложности каждого следующего уровня вложенности, аналитическое выражение сложности S по Мак-Кейбу будет следующим:

$$S_M = 1 + \sum_{i=0}^{k} 2^i N_{i\_if,wh,for} + N_{size>50} + N_{and,or} \qquad (8)$$

где i -уровень вложения условных и циклических операторов, N_i - количество различных операторов i -го уровня, $N_{size>50}$ - количество операторов циклов и условий длиной более 50 строк.

Данная метрика была предложена Мак-Кейбом еще в 1976 году. Несмотря на прошедшие 35 лет, она сохраняет свою актуальность, поскольку точнее выражает сложность фундаментальных структур ПО, в отличие от равнозначных операторов Холстеда. Автор метрики отмечает высокую корреляцию метрики с ошибками. Поэтому использование данной метрики для оценки ошибок обоснованно и логично.

Наиболее известной метрикой, выражающей сложность структур данных программы, является **метрика Чепина** [13]. Данная метрика анализирует характер используемых переменных и оценивает поток данных количественно в виде выражения:

$$S_C = P + 2M + 3C + 0,5T \qquad (9)$$

где P – не модифицируемые переменные, M – модифицируемые переменные, C – управляющие переменные, T – не используемые переменные. Поскольку современные системы программирования не ограничиваются только переменными, данную метрику необходимо дополнить элементами массивов, перечислений и других структур данных, а также псевдонимами, указателями и полями таблиц баз данных, если таковые используются в ПО. При дополнении необходимо использовать классификацию весовые коэффициенты, назначенные автором. С учетом предложенных дополнений метрика точнее оценит поток данных в программе, и может быть использована для оценки ошибок.

Немаловажной компонентой сложности ПО является межмодульная связность (связность методов, классов). В [11] модульную сложность ПО описывает **метрика Джилба** в виде абсолютного N_{zv} и относительного количества межмодульных связей:

$$N_{zv\_mod} = \frac{N^4_{zv}}{N_{mod}}, \qquad (10)$$

где N_{mod} -количество модулей ПО. Чем выше показатель модульной связности, тем сложнее проект и большее число ошибок.

Данная метрика очень актуальна для современного ПО с высокой степенью сопряжения модулей, классов, методов.

В отношении метрики, оценивающей сложность вычислений в программе, необходимо отметить, что среди большого разнообразия метрик эта метрика не была обнаружена. Поскольку некоторая часть программных ошибок связана с различными вычислениями и преобразованиями, необходима разработка данной метрики и ее учет для оценки ошибок.

Результаты подсчета метрик сложности согласно выражению (7) для семи программных проектов приведены в табл. 3.

Таблица 3

Экспериментальные результаты подсчета метрик сложности.

Метрика	Проект1	Проект2	Проект3	Проект4	Проект5	Проект6	Проект7	Коэф. коррел. метрик и факт. дефектов
Размер кода	18	20	240	184	543	523	553	0,9321
Число строк кода	677	791	9417	6840	20653	20037	22164	0,9408
Число модулей	6	9	43	41	74	74	76	0,9452
Число таблиц БД	2	2	15	14	16	24	17	0,8427
Число полей БД	38	30	292	211	321	460	325	0,8509
Фактическое число дефектов	4	6	36	39	57	76	89	
Метрика Мак-Кейба	58	120	533	601	792	819	1123	0,9807
Метрика Чепина	157	128	2298	1274	2399	2246	3242	0,9214
Метрика Джилба N_{zv}	8	5	126	86	225	278	276	0,9746

Анализируя полученные результаты, можно сделать следующие выводы:

- абсолютные значения данных метрик количественно характеризуют управляющую логику, поток данных и межмодульную связность ПО;

- разделив абсолютные значения метрик на число строк исходного кода, мы получим относительную величину каждой метрики, по которой проекты можно сравнивать между собой по разным аспектам сложности;

- коэффициент корреляции (КК) метрик сложности выше КК размерных метрик, т.о. оценка ошибок на базе метрик сложности будет точнее;

- метрика Мак-Кейба имеет наивысший КК-**0,98**. Поэтому сложность управляющей логики – наиболее значимый фактор возникновения ошибок в данных проектах;

- метрика Джилба с КК **0,97** характеризует межмодульную связность, как второй по значимости фактор возникновения ошибок в данных проектах;

- метрика Чепина с КК **0,92** говорит о том, что поток данных программы наименее связан с числом дефектов в данных проектах.

4. Метод оценки программных дефектов

Метод оценки программных дефектов на базе метрик сложности предполагает два подхода:

Использование единой комплексной метрики, объединяющей различные метрики сложности, и общего числа дефектов ПО;

Использование различных метрик сложности, и различного числа дефектов, относящихся к различным видам сложности.

Проблема реализации 1-го подхода заключается в использовании актуальной комплексной меры сложности. Наиболее известная мера Кокола [11] была предложена исследователем около 30 лет назад, основывалась на метриках Холстеда и LOC-оценке, показывающих большие отклонения и низ-

кую корреляцию с количеством дефектов в табл. 1 и табл. 2.

2-й подход основан на соотношении числа наблюдаемых дефектов и конкретного вида сложности в виде значения определенной метрики. Например, число дефектов в операторах циклов и условий связаны со сложность управляющей логики ПО в виде значения метрики Мак-Кейба.

Метод на основе 2-ого подхода состоит из следующих этапов:

1. Выполнить **эмпирическое наблюдение дефектов** конкретного ПО отнесением каждого выявленного дефекта к определенному типу дефектов $N_{emp} = \{N_{1,}, ... N_k\}$, установить соответствие между типами дефектов и видами сложности данного ПО;

2. Из множества известных метрик $M_{all} = \{M_1, ..., M_n\}$ аналитически **выбрать метрики,** наиболее подходящие для количественного подсчета различных видов сложности данного ПО: $M_{sel} = \{M_1, ... M_k\}$, $M_{sel} \in M_{all}$;

3. С помощью анализаторов исходного кода ПО **подсчитать** значения выбранных **метрик** $S = \{S_{1,}, ... S_k\}$;

4. Рассчитать значение **дефектосложности** D_i размерностью количество единиц сложности на 1 дефект для каждой метрики

$$D_i = \frac{S_i}{N_i} \forall D_i \in D = \{D_1, ... D_k\} \qquad (11)$$

с целью применения полученных значений дефектосложности для последующей оценки дефектов;

5. Для прогнозной оценки дефектов при разработке или изменении ПО **подсчет новых** значений **метрик** сложности $S^*_{new} = \{S_{1,}{}^*, ... S_k{}^*\}$;

6. **Расчет** нового числа дефектов по новым значениям метрик и полученной ранее дефектосложности для каждой метрики;

$$N^*_i = \frac{S^*_i}{D_i} \forall N^*_i \in N^*_{new} = \{N^*_1, ... N^*_k\}. \qquad (12)$$

7. **Оценка** общего количества дефектов

$$N^*_{sum} = \sum_{i=1}^{k} N_i^*. \qquad (13)$$

Т.о. данный метод позволяет на основании данных одного проекта, выполнить прогнозную оценку дефектов при изменении данного или разработке других проектов, еще до начала тестирования. Затем разница между прогнозной оценкой дефектов и числом обнаруженных при тестировании составит количество оставшихся в коде дефектов, и позволит оценить показатели эксплуатационной надежности ПО.

Для проверки точности метода оценки дефектов использовались метрики Мак-Кейба и Джилба с КК>0,97 и экспериментальные данные таб. 3.

Дефектосложность в проекте 1 согласно (11) по Мак-Кейбу составила 20 единиц, по Чепину – 8 единиц сложности на 1 ошибку.

Результаты расчетов согласно (12) и (13) приведены в табл. 4.

Таблица 4

Результаты расчетной оценки ошибок

Показатели	П.2	П.3	П.4	П.5	П.6	П.7
$N_1{}^*$ по м. МакКейба	6	27	30	40	41	56
$N_2{}^*$ по м, Джилба	1	15	11	28	35	36
N^*_{sum}	7	42	41	68	76	91
N_{emp}	6	36	39	57	76	89
% отклонений	10%	18%	5%	19%	0%	2%

Наибольшее отклонение составило 19%, среднее - 9%. **Точность оценки выше** в 4,58 раза точности оценки дефектов по Холстеду в табл. 2.

Преимуществом данного метода является **более точная оценка** дефектов по сравнению с моделями Холстеда, Акиямы и Гафнии. Поскольку метрики сложности оценивают свойства ПО не зависимо от языков программирования, **данный метод применим для различных языков** и средств разработки. Необходимые исходные данные для реализации метода – исходный текст ПО с известным числом дефектов. Необходимое средство для реализации метода - анализатор программного кода для автоматизированного подсчета метрик.

Литература

1. Ostrand, T. Collecting and Categorizing Software Error Data in an Industrial Environment [Text] / T. Ostrand, E. Weyuker // Journal of Systems and Software. – 1984. – № 4 (November). – P. 289 – 300.

2. Balzert, H. Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering [Text] / H. Balzert. – Spektrum Akademischer Verlag Heidelberg, 2009. – 624 p.

3. Dijkstra, E. On the Cruelty of Really Teaching [Text] / E. Dijkstra // Computer Science.» Communications of the ACM 32. – 1989. – No. 12 (December). – P.

1397-1414.

4. Макконнелл, С. Совершенный код. Мастер-класс [Текст]: пер. с англ. / С. Макконнелл. – М. : Издательско-торговый дом «Русская Редакция»; СПб.: Питер, 2005. – 896 с.

5. IEEE Standart Glossary of Software Engineering Terminology [Text] / IEEE Std 610.12-1990.

6. Харченко, В.С. Оценка и обеспечение качества программных средств космических систем [Text] / В.С. Харченко, В.В. Скляр, Б.М. Конорев. –Х.: НАУ «ХАИ», 2007. – 244 с.

7. Coverity scan [Электронный ресурс]. – Режим доступа: http://scan.coverity.com. – 23.01.2012.

8. Lyu, M.R. Handbook of Software Reliability Engineering [Text] / M.R. Lyu. – McGraw-Hill Company, 1996. – 805 p.

9. Gaffney, J.R. Estimating the Number of Faults in Code [Text] / J.R. Gaffney // IEEE Trans. Software Eng. – 1984. – Vol. 10 (4). – P. 357 – 362.

10. Холстед, М.Х. Начала науки о программах [Текст]: пер. с англ. / М.Х. Холстед. – М.: Финансы и статистика, 1981. – 128 с.

11. Поморова, О.В. Інтелектуальний метод оцінювання результатів проектування та прогнозування характеристик якості програмного забезпечення [Текст] / О.В. Поморова, Т.О. Говорущенко // Радіоелектронні і комп'ютері системи. – 2010. – № 6 (47). – С. 211-218.

12. McCabe, T.A. Complexity Measure [Text] / T.A. McCabe // IEEE Transaction on Software Engineering. – 1976. – 4, № SE-2. – P. 308 – 320.

13. Программный код и его метрики [Электронный ресурс]. – Режим доступа: http://habrahabr.ru. – 23.01.2012.

Поступила в редакцию 23.01.2012

Рецензент: д-р техн. наук, проф. О.В. Поморова, Хмельницкий национальный университет, Украина.

МЕТОД ОЦІНКИ КІЛЬКОСТІ ПРОГРАМНИХ ДЕФЕКТІВ З ВИКОРИСТАННЯМ МЕТРИК СКЛАДНОСТІ

С.О. Яремчук

В роботі виконано критичний аналіз найчастіше використовуваних метрик програмного забезпечення. Визначена ступінь можливості використання метрик розміру і складності для оцінки кількості програмних дефектів. Для більш точної оцінки метрики Мак-Кейба і Чепіна доповнено раніше не врахованими аспектами програмної складності. Виявлено відсутність метрики складності обчислень. Поведені підрахунки метрик для семи програмних проектів. Запропоновано метод оцінки кількості програмних дефектів з використанням метрик складності. Для перевірки методу виконано розрахунки оцінки дефектів, визначена точність оцінок методу, вказані його переваги.

Ключові слова: складність програм, дефекти програмного забезпечення, метрики розміру, метрики складності, метод оцінки кількості програмних дефектів.

THE METHOD OF EVALUATION OF THE NUMBER OF SOFTWARE DEFECTS USING COMPLEXITY METRICS

S. A. Yaremchuk

The work includes critical analysis of the most frequently used software metrics. The degree of possibility to use size and complexity metrics for the evaluation of software defects number was defined. For more precise evaluation of complexity, McCabe's and Chapin's metrics were supplemented by software complexity aspects, which were not specified before. The lack of calculation complexity metric was detected. The metrics for 7 software projects were calculated. The method of calculating the number of software defects using complexity metrics was suggested. The method was verified by the calculation of defect evaluation, the precision of the method evaluations, its advantages were shown.

Keywords: software complexity, software defect, size metrics, complexity metrics, the method of evaluation of the software defects.

Яремчук Светлана Александровна – аспирант Одесского национального политехнического университета, Одесса, Украина, старший преподаватель кафедры информационных систем и технологий Измаильского института водного транспорта, Измаил, Одесская область, Украина, e-mail: svetlana397@yandex.ru.