УДК 004.412+004.413.4

В. О. Мищенко, *д.т.н., профессор,*
Харьковский национальный университет имени В. Н. Каразина,
пл. Свободы, 4, г. Харьков, Украина
mischenko@univer.kharkov.ua
С. М. Первунинский, *д.т.н., профессор,*
Черкасский государственный технологический университет,
б-р Шевченко, 460, г. Черкассы, 18006, Украина
cherkpervun@rambler.ru

ТОЧНОСТЬ ОЦЕНКИ СТАТИЧЕСКИХ МЕТРИК ЭНЕРГЕТИЧЕСКОГО АНАЛИЗА ПРОГРАММ

Статья посвящена оценке степени определённости числовых атрибутов энергетических метрик компьютерных программ и вопросам точности выводов о трудности модулей программ, основанных на данных метриках. Речь идёт о статических метриках процесса разработки программных систем, примитивы которых восходят к метрикам Холстеда. Показано, с привлечением данных о модулях программ, разработанных на разных языках, каким образом риски ошибочных выводов, вытекающие из технической неточности оценок энергетических метрик, указанными способами могут быть сделаны достаточно малыми в контексте прогностического характера выводов, основанных на этих метриках.

Ключевые слова: *качество программ, надёжность, метрики, примитивные характеристики, энергетический анализ программ, риск, релевантность, точность, достоверность.*

Постановка проблемы и цель работы. Энергетический подход к статическому анализу программного обеспечения (ПО), очень кратко, состоит в следующем [1–3]. Программные системы (ПС) и их модули, состоящие из многих лексических единиц, могут оцениваться (подобно системам, состоящим из молекул и атомов) с помощью интегральных характеристик. На законченных этапах разработки исходных текстов ПС использованные в этих текстах лексические единицы выступают в качестве средства реализации компонентов и интерфейсов. Отношения между компонентами, члены классов и элементы всевозможных интерфейсов также могут быть представлены характеристиками, которые, отражают архитектурные решения. Их смысл имеет общие черты с внутренней энергией физических систем. Сопоставление характеристик архитектуры и лексики, в свою очередь, порождают характеристики, которые позволяют, в общем и целом, судить о процессе разработки как воплощении архитектурных решений в программном коде. Это напоминает сопоставление разных способов передачи энергии в физике. Этот подход для современных сложных программных систем использует метрики энергетического анализа. Они образуют собственную систему, свои методы, но используют примитивы, которые были введены в практику анализа качества программ М. Холстедом [4]. Метрики и методы Холстеда относилась, по сути, к простым программам, одномодульным, структурно однородным, не содержавшим никаких описаний. Однако идеи физических аналогий теории Холстеда послужили отправной базой создания энергетического анализа программ [2, 3].

Энергетический анализ программ относится к сфере контроля качества ПС в аспекте надёжности [5], позволяя судить о трудности отдельных модулей и зрелости системы в целом [2, 3]. С его помощью были решены актуальные исследовательские и технические задачи [6–9]. Следующий шаг развития – создание информационных технологий. Это требует исследования условий, при которых технические неточности оценки метрик не несут угрозы ошибочных суждений о попадании результата оценок в те или иные рамки. Этой задаче посвящена данная работа.

Её **цель** – изучение рисков достоверности суждений энергетического анализа в связи с возможными неточностями в оценках используемых метрик.

Отметим, что будут рассматриваться только ПС, которые проектируются и разра-

батываются человеком. Иные направленности возможных исследований приведены в [10].

Описание проблемы и постановка задачи. Обзор известных результатов начнём с некоторых метрик М. Холстеда [4]:

$$V = V(N,\eta) = N \log_2 \eta\,, \qquad (1)$$

где V – определяемый данной формулой *объём* (volume) модуля программы;

η – *словарь* (vocabulary) модуля программы – число разных *программных символов* (*ПСим*), использованных в исходном тексте этого модуля (ПСим – это одна или несколько лексем данного языка программирования, неразрывно связанных по смыслу);

N – *длина* (length) модуля программы – общее число программных символов (с учётом повторений).

Число ошибок кодирования:

$$B = V / E_0 \quad (E_0 = 3000)\,; \qquad (2)$$

Потенциальный объём:

$$V^* = V(\eta^*,\eta^*) = \eta^* \log_2 \eta^*, \qquad (3)$$

где $\eta^* = 2 + \eta_2{}^*$ – *потенциальный словарь* (potential vocabulary) модуля программы, а

$\eta_2{}^*$ – *число параметров ввода–вывода*. Последний параметр имел лишь нестрогое пояснение [4] и от этого примитива к 90-м гг. отказались почти все практики.

Трудность модуля:

$$D = V/V^*. \qquad (4)$$

Вместо неё, ввиду проблем с примитивом η^* и, соответственно, с метрикой V^*, обычно использовалась метрика *оценки трудности*:

$$\hat{D} = (\eta_1 / \eta_1{}^*)\cdot(N_2 / \eta_2) = \qquad (\eta_1{}^* = 2), \quad (5)$$

где N_2 – *число операндов*, т.е ПСим в тексте модуля, имеющих содержательное назначение;

η_2 – *словарь операндов*;

η_1 – *словарь операторов* (т.е. управляющих, функциональных ПСим);

$\eta_1{}^*$ – *словарь операторов потенциальной реализации* (равенство $\eta^*_1 = 2$ является, по сути, соглашением о нормировке).

Со временем от (4) отказались, а метрику (5) *де факто переименовали* в трудность [11]. В отношении многомодульных программ это неприемлемо [12, 3].

Вследствие проблем с $\eta_2{}^*$ ключевая метрика Холстеда – *усилие* программирования – имела одновременно концептуальную трактовку

$$A = D\cdot V \qquad (6)$$

и практическое определение

$$\hat{A} = \hat{D}\cdot V. \qquad (7)$$

Достижения холстедовской системы мер качества таковы: практическая эффективность диаграмм Парето, которые строятся на основе оценки трудности модулей; анализ тенденций стоимости разработки программ при использовании разных (требующих сравнения) технологий на основе сравнения метрик оценки усилия; возможность (иногда – единственная) получить данные для оценки рисков, связанных с остаточными ошибками на основе тех или иных математических моделей надёжности ПО (при перечислении средств контроля качества мы придерживаемся терминологии [5]). Однако в последние 10-15 лет сообщения о применении метрик Холстеда встречаются редко, в основном в связи с исследовательскими проектами (подробнее в [2]), а современные руководства по созданию надёжного ПО критикуют их за отсутствие стандартизации [11, 13].

Достоинства и проблемы холстедовских метрик послужили ориентирами при разработке *метрик энергетического анализа*. Во-первых, метрика объёма (1) обобщена так, чтобы учесть ассоциацию модуля ПС с другими, от которых он зависит в смысле импорта описаний, а также учесть порядок разработки модулей. Обобщенная метрика – *объём разработки* модуля – даётся выражением вида

$$W = V(N, \eta + \Delta\eta) + \sum_k V_k\,, \qquad (8)$$

где $V(N,\eta)$ – объём (2) оцениваемого модуля как функция длины и словаря (1);

V_k – холстедовские объёмы тех интерфейсных модулей, от которых данный модуль зависит и которые разработаны раньше него;

$\eta + \Delta\eta \geq \eta$ – «эффективное» значение словаря данного модуля, которое имеет поправку, отражающую наличие отношений реализации и наследования, а также запаздывание разработки этого модуля с точки зрения логичной последовательности создания данной ПС (детали определения в [2–3].

Из сказанного очевидна необходимость точных определений: модуля, интерфейсного модуля и различных отношений между модулями. Более того, следующая метрика будет требовать определения блоков и их групп, которые позволяют описывать внутреннюю

архитектуру модулей. Целям такого описания служат аксиомы о *схемах программных систем* (СПС) [1]. Для языков программирования (ЯП) с хорошо развитыми средствами компонентного и объектно-ориентированного программирования (ООП) достаточно один раз произвести обоснованную интерпретацию СПС в данном языке для того, чтобы в дальнейшем в программах на этом языке можно было однозначно идентифицировать модули, группы, блоки и отношения между ними в смысле СПС. В энергетическом анализе метрика (3) получила полноценную прописку, поскольку «словарь операндов потенциальной реализации» η_2 * имеет в СПС точный смысл числа формальных параметров блока, но он является характеристикой не модуля, а блока. При подсчёте этой величины к числу формальных параметров вызова (если они у блока есть) прибавляется число параметров настройки этого блока (если они есть) и число «квазиобъектов ввода-вывода», которые однозначно подсчитываются по программному коду тела блока [2, 4]. Последнее предполагает, что известны операции, которые относятся ко вводу и выводу. На этом пути потенциальный объём (3) формализуется, а метрика *трудности разработки* модуля определяется как

$$D = W/V^*. \qquad (9)$$

Теперь и выражение вида (6) определяет корректную метрику, названную работой программирования. В энергетическом анализе вводятся также новые метрики. Первая из них использует вид зависимости, к которой пришел Холстед, пытаясь «аппроксимировать» свою метрику усилия (6) для целей прогноза. Попытка была неудачной (подробно об этом в [3]), но подобранная им кубическая нелинейность оказалась востребованной в энергетическом анализе при построении аналога внутренней энергии.

Метрика *спецификационная энергия* СПС [1, 2]:

$$E = \sum_i E_i \text{ (для ПС, сумма по модулям)},$$

$$E_i = \sum_g E_{i,g} \text{ (для модуля)},$$

$$E_{i,g} = \left(\sum_b V^*_{i,g,b} \right)^3 \Big/ \lambda^2_{i,g} \quad (g \neq 0)$$

(для структурной группы модуля), (10)

$$E_{i,0} = \sum_b \left(V^*_{i,0,b} \right)^3 \Big/ \lambda^2_{i,0,b}$$

(для автономной группы), (11)

где i, g, b – индексы, соответствующие *интерфейсным* модулям, *интерфейсным* группам внутри модуля и *интерфейсным* блокам внутри групп, причём 0-й группой является т.н. автономная группа (то, что остаётся от модуля после изъятия из него других внутренних явно определённых групп), которая всегда содержит, по крайней мере, т.н. остаточный блок («заголовочную» часть модуля);

$\lambda_{i,g}, \lambda_{i,0,b}, \lambda_i$ – значения уровня языка программирования *i*-го модуля, возможно, разные для разных модулей, и, возможно, разные (но редко) для разных групп или блоков.

Детали выделения групп и методы оценивания спецификационной энергии групп, отличных от автономной и структурной, имеются в [17, 3, 4]. Уровни ЯП оценивают непосредственно по схеме Холстеда [4] или косвенно [2].

Метрика *интеллектуального тепла*

$$Q = E - A \qquad (12)$$

имеет смысл для самостоятельных модулей, подсистем и ПС в целом и позволяет сроить схему контроля качества процесса разработки по аспектам управляемости (предсказуемость оценок) и зрелости разработки (стабилизация оценок). Также эта метрика (в модифицированной форме) позволяет судить о приемлемости баланса между материализованным в архитектуре системы опытом разработчиков и затратами труда на кодирование, а также следить за контрольными пределами вариаций интеллектуального тепла в разработке. Модифицированная форма

$$q = Q/\max(A, E) \qquad (13)$$

оценивает, грубо говоря, ментальные действия при разработке ПС, которые как бы формируют ($q < 0$) или передают ($q > 0$) опыт разработчика в программный код. Зрелая программа несёт признак сбалансированности этих процессов (малость q).

Выгоды энергетического анализа, помимо обобщения метрик объёма и трудности, которые становятся применимыми в анализе качества современных ПС, позволяют отслеживать в процессе разработки признаки зрелости системы [2].

Постановка задачи работы: выявить источники риска, связанные с приближенным характером оценок энергетических метрик; оценить наиболее значительные риски; выработать методы реагирования на эти риски.

Основной материал исследования и полученные результаты. Всюду в дальнейшем предполагаем, что оценка проводится для ПС или их подсистем, для которых энергетические метрики валидны. Например, в модулях, существенно превосходящих другие по объёму разработки (8), при выборочной проверке должно оказаться больше дефектов, вызванных ошибками кодирования.

При фильтрации модулей проекта с целью выявить наиболее проблемные из них в энергетическом анализе используются три метода, которые основаны, соответственно, на метрике оценки трудности (5), на метрике трудности (9) и на признаке зрелости по степени энергетической сбалансированности (близости значений (13) к нулю) [3]. Каждый из подходов является прогностическим и использует далеко не всю содержательную информацию, которую несёт программный код. Поэтому для получения доверительных результатов используются, по возможности, все три метода. Они нацелены на разное (первый «судит» по размеру модуля, учитывая также разнообразие использованных лексических средств, второй соотносит размеры формальной спецификации и реализации, а третий, в определённом аспекте, оценивает совершенство стиля разработки). Однако, если точность оценки примитивов низкая, то у этих метрик могут появиться значительные случайные отклонения, так что фильтрация даже на основе трёх методов станет ненадёжной.

Рассмотрим угрозы точности оценки примитивов. Для V – это нечёткость трактовки ПСим, для W – точность поправки на отношения с другими модулями, для $D\hat{}$ (и только для этой метрики) – это неправильный учёт операторов и операндов, а для $V*$ (и D) – нетипичные интерфейсы, для Q, q – нереалистичная оценка уровня ЯП.

Методы расчета длины и словаря принято называть *стратегий подсчёта* (counting strategy). Общего правила формирования такой стратегии не существует. Чем выше уровень языка, тем сложнее конструкции, которые можно признать его смысловыми атомами, но тогда становятся возможными исключающие друг друга варианты, выбор из которых субъективен. Однако, даже если для одного и того же языка используются разные стратегии подсчёта, результаты оценки объёмов (1) могут различаться с практической точки зрения незначительно [10]. К тому же, неопределённость в значении объёма менее 1500 сим·бит, как видно из (2), не скажется на прогнозе возможного числа ошибок (3), если оно округляется до целых.

Относительная погрешность оценки объёма прямо соответствует относительной ошибке измерения N, а при $32 < \eta < 256$ величина $\log\eta$ варьируется всего лишь между 5 и 8. Приведём экспериментальные данные о влиянии различий в подсчёте длины и словаря для двух различных языков. Для языка Java в [9] рассматривались (с иной точки зрения, чем в данной статье) три стратегии подсчёта (одна реализовывалась приложением, представленным на сайте [15], другая – студенческой программой, третья – другой студенческой программой, воплотившей подход [2]). Введём меру расхождения двух оценок как

$$\delta V = \left|V - V'\right| / V, \qquad (14)$$

где V – объём модуля, оценённый по той стратегии, которая априори считалась точнее;

V' – объём того же модуля, оценённый по второй стратегии.

Обследовалось около десятка ПС, разработанных на языке Java для мобильных устройств (в среднем эти системы содержали примерно по десятку модулей). На основе выборочного метода получен такой результат:

для 20 % модулей, оценка которых не нуждалась в согласовании методов, среднее расхождение между любыми двумя использованными методами можно считать равным 0,033 при стандартном отклонении 0,019, так что в большинстве случаев расхождение не превосходит 0,042, а максимальное ожидаемое расхождение оценивается как 0,080;

для остальных 80 % модулей, для которых следовало бы согласовать специальные ситуации оценивания по разным методам, среднее расхождение между двумя методами оценивается как 0,28 при стандартном отклонении 0,53, так что ожидаемые значения расхождения (2) могут достигать до 2,0 (фактически встречалось значение 1,30).

Здесь под специальными ситуациями понимается реализация того риска, что авторы разных методов могут по-разному трактo-

вать омонимы и комментарии в программе. Визуальный просмотр модулей, вызвавших расхождения более, чем на 10 % (как правило, между всеми методами), подтвердил решающий вклад подобных специальных ситуаций.

Другой эксперимент проводился с выборкой 101 модуля Ада программ, которые случайно отбирались в почти равных долях из репозитория учебных программ, из библиотек для бизнес-приложений и в ПС прикладного назначения. Использовались реализации стратегий разных авторов по методам [2] и [16]. По специальным ситуациям методы были полностью согласованы. Результат такой:

среднее значение δV (2) составило 0,078 при стандартном отклонении 0,014, так что в большинстве случаев расхождение не превосходило 0.092, а максимальное ожидаемое расхождение оценивается как 0,12.

Вывод состоит в том, что реальная угроза определённости или точности оценки объёма модуля программы заключается в несоответствии правил стратегий подсчёта, определяющих учёт комментариев и омонимов в программе. Эта угроза снимается согласованием для каждого ЯП применяемых стратегий подсчёта по специальным ситуациям.

Рассмотрим теперь оценку объёма разработки (8). Превышение этой величины над холстедовским объёмом того же модуля, как показано в [3, 4], происходит в основном за счёт прибавления объёмов тех модулей, от которых зависит данный. Хотя это не так для модулей с запаздывающей разработкой, их относительный вклад в характеристики проекта обычно мал [4], так что этот специальный случай мы опустим (там серьёзных угроз точности оценки не возникает). В силу этого относительная погрешность оценки не может превосходить рубеж, который ограничивает относительные погрешности оценки холстедовских объёмов. Поэтому вывод относительно точности оценок холстедовских объёмов V сохраняется для объёмов разработки W.

Переходя к (5), перепишем формулу:

$$D\hat{} = (\vartheta^{-1} - 1) \cdot \Theta \cdot \frac{N}{2}, \qquad (15)$$

где $\vartheta = \eta_2 / \eta$; $\Theta = N_2 / N$.

Характер распределения безразмерного параметра Θ получим из опыта. Для языков низкого уровня используем данные [4, с. 40] о примитивах кода с исполняемыми операторами без описаний (старый Алгол), что характерно для языков ассемблерного уровня:

Центр (Θ)=0,483 (объём выб.=20);

Полуразмах (Θ)=0,067 (знач. 5%). (16)

С другой стороны, по выборке всех модулей современной объектно-ориентированной прикладной программы на языке высокого уровня (ЯВУ) C++, разработанной с нашим участием, получено:

Центр (Θ)=0,700 (объём выб.=45);

Полуразмах (Θ)=0,173 (знач. 5%). (17)

В обоих случаях эмпирическое распределение можно было считать равномерным. Характер распределения безразмерного параметра ϑ иной. Наша оценка по данным [4, с. 35] даёт:

Среднее (ϑ)=0,611; Станд. отклон.(ϑ)=0,156;

оценка Мин.($\vartheta(1-\vartheta)$) = 0.100, (18)

($\vartheta(1-\vartheta)$ имеет плохо идентифицируемое J-образное распределение, и приведенная для неё оценка эмпирическая). Для модулей программы распределение ϑ другое, имеет признаки нормального:

Среднее (ϑ)=0,405; Станд. отклон.(ϑ)=0,152;

оценка Мин. ($\vartheta(1-\vartheta)$) = 0.143 (19)

(оценка минимума не хуже 5 % значимости). При этом сомнения по поводу отнесения ПСим к операндам или операторам и отождествления омонимов возникали редко (менее 0,5 % случаев), поскольку язык *C++* базируется на родном синтаксисе языка *C*, значительно более низкого уровня, чем он сам. Если же ЯВУ имеет более сложный синтаксис, то указанные проблемы могут стать непреодолимыми. Пример – язык Ада [10].

Оценку относительной погрешности вычисления (12) делаем на основании выражения для дифференциала логарифма:

$$\delta D\hat{} = d\ln(D\hat{}) = \frac{d\Theta}{\Theta} - \frac{d\vartheta}{\vartheta(1-\vartheta)},$$

$$|\delta D\hat{}| \le \frac{\max|d\Theta|}{\min\Theta} + \frac{\max|d\vartheta|}{\min(\vartheta(1-\vartheta))}. \qquad (20)$$

Полагаясь на оценки (13)–(16), получим:

$|\delta D\hat{}| \le |d\Theta| / 0{,}414 + |d\vartheta| / 0{,}100$ (для языков низкого уровня); (21)

$|\delta D\hat{}| \le |d\Theta| / 0{,}572 + |d\vartheta| / 0{,}143$ (для ЯВУ). (22)

В силу уже сказанного о проблеме точности дифференциации ПСим на операторы и операнды, вытекающей из неформального характера этих понятий и обостряющейся для языков высокого уровня, необходимо оценить последствия того или иного уровня точности, которые возможны на практике. Эти уровни должны назначаться относительно характерных мер отклонений от средних для рассматриваемых параметров. Для относительного числа операндов (словарей операндов) такие отклонения, отражающие *особенности* модуля, у нас не превосходили 0,25 (0,3), так что отклонения такой же величины за счёт *погрешности измерения* не позволяют говорить о точности вообще. Рассмотрим в качестве ориентира тест $N = 64,\ N_2 = 32,$ $\eta = 24, \eta_2 = 12$ (это около 10 строк кода, модули с заметно меньшими значениями сами по себе практически не интересуют программометриста). Введём четыре класса точности прогрессивно и так, чтобы при средней точности по Θ и ϑ допускалась абсолютная погрешность хотя бы в 1 ПСим, а при высокой – уже нет. Эти условия однозначно определяют рубежи классов точности:

крайне низкая точность:

$$0{,}125 \le |d\Theta| < 0{,}25,\quad 0{,}2 \le |d\vartheta| < 0{,}3; \quad (23)$$

невысокая точность:

$$0{,}0625 \le |d\Theta| < 0{,}125,\ 0{,}1333 \le |d\vartheta| < 0{,}2; \quad (24)$$

средняя точность: $0{,}0301 \le |d\Theta| < 0{,}0625,$

$$0{,}0889 \le |d\vartheta| < 0{,}1333; \quad (25)$$

высокая точность:

$$0{,}0 \le 0|d\Theta| < 0{,}0301,\ 0{,}0 \le |d\vartheta| < 0{,}0889 \quad (26)$$

(подразумевается, что в каждом классе условие на один из параметров выполняется точно, а второе – тоже точно либо нарушено в сторону более высокой точности). Значение оценки трудности оценивается с точки зрения попадания в один из шести классов [14], образуемых на положительной полуоси границами:

$$D_1 = 115,\ D_2 = 160,\ D_3 = 250,\ D_4 = 340,\ D_5 = 430. \quad (27)$$

(в 0-й класс трудности попадают модули, у которых $D\hat{} \le D_1$, а в 5-й – модули с $D\hat{} > D_5$).

Под *индексом занижения* (оценки трудности) будем понимать индекс того «наименьшего» класса трудности, в который может попасть ненормально трудный модуль (класс 5), оценённый с данным уровнем точности. Индекс занижения = 4 – нормальный, а = 0 – наихудший. Под *индексом завышения* будем понимать индекс того «наибольшего» класса трудности, в который может попасть лёгкий модуль (класс 0), оценённый с данным уровнем точности (индекс завышения = 1 – нормальный, а 5 – наихудший). Метод состоит в следующем. Пусть оценка трудности модуля лежит в 5-м классе у его границы D_5. Пусть погрешности параметров, допущенные при оценивании, при заданном уровне точности (из (23)–(26)) таковы, что неравенства в (21)–(23) сколь угодно близки к равенствам, а $\delta D\hat{} < 0$. Тогда, вычитая из D_5 соответствующую абсолютную погрешность и сравнивая полученное значение с границами (27), узнаем, в какой класс может ошибочно попасть оценка в худшем случае (индекс занижения). Аналогично определятся индекс завышения. Все эти индексы указаны в табл. 1, 2.

Таблица 1

Зависимость точности попадания в классы трудности от оценок параметров Θ, ϑ на заданном уровне точности для языков низкого уровня

Уровень точности оценки Θ, ϑ	Завышение индекса	Занижение индекса
высокая точность	2	2
средняя точность	0	2
невысокая точность	0	2
крайне низкая точность	0	3

Переходя к анализу и выводам по точности оценки метрики $D\hat{}$ (5), подчеркнём, что таблицы указывают лишь на худшие возможные, однако маловероятные, результаты ошибки практического определения класса трудности по данной метрике.

Таблица 2

Точность попадания в классы трудности в зависимости от уровня точности оценок параметров Θ, ϑ для языков высокого уровня

Уровень точности оценки Θ, ϑ	Занижение индекса	Завышение индекса
высокая точность	3	1
средняя точность	2	2
невысокая точность	0	2
крайне низкая точность	0	3

При случайных величинах и погрешностях их определения оценка, как правило, остаётся в «своём» классе. Однако с точки зрения разработки ПС для критических применений индексы завышения в табл. 1, допускающие, в принципе, ошибочное причисление к лёгким даже «ненормально трудных» модулей (класса 5), свидетельствуют о значимом риске для оценки кода, разработанного на языках низкого уровня. Остальные результаты таблиц можно считать удовлетворительными, предполагая, что высокая или средняя точность оценок достижимы при приемлемых затратах. Выявленный риск даёт объяснение некоторым известным фактам:

1) несмотря на то, что метрика $D^{\wedge}$ десятки лет успешно применялась программной индустрией, документ [13] не рекомендует её в качестве стандартного средства контроля критического ПО на основании того, что она для успешного применения требует более высокой квалификации персонала в области прикладных статистических методов, чем ожидается. Действительно, мы показали, что, если не обеспечивать всегда высокую (23) точность оценивания, то выводы по оценке метрики (5), в принципе, могут оказаться недостоверными;

2) отклонения $d\Theta, d\vartheta$ могут возникать не только вследствие измерений, но и вследствие частых вариаций исходных кодов в незрелой ПС. С этой точки зрения табл.1, 2 объясняют, почему классы трудности по метрикам (4) и (5) отлично соответствуют друг другу в зрелой (давно устоявшейся по исходным кодам) ПС, тогда как в незрелой они перемешаны [14].

Есть способ парировать угрозы реализации обсуждаемого риска. Он требует одноразовых дополнительных затрат на изучение статистики проектов и состоит в том, чтобы вместо вычисления параметров Θ и ϑ по нестандартизованным эмпирическим правилам использовать для модулей на данном языке и относящихся к данной области применений вменённые значения этих параметров. Только так, например, удаётся практически использовать метрику $D^{\wedge}$ при анализе программ на языке Ада [6–7, 14].

Обычно ошибка подсчёта работы программирования (эта работа пропорциональна квадрату объёма) приемлема, если не превышает 20 %. Следовательно, при наличии проблем с измерениями можно соглашаться на 10 %-ю ошибку в подсчёте объёмов программ ($\sqrt{1.2}\approx 1.1$). Выше мы видели, как её обеспечить. Остаются метрики (10)–(11). Причём последние две зависят от первой из них «почти линейно». Остаётся выяснить, можно ли гарантировать приемлемую точность оценки (10)? Очевидно, что величина её оценки сильно чувствительна к точности оценки V^*. Для достижения 10 % точности оценки E потенциальные объёмы следовало бы оценивать практически абсолютно точно. Выход подсказывает то, что эта метрика чувствительна в той же степени к вариациям в исходных текстах, как и к погрешностям измерений. Ввиду этого потенциальные энергии (а с ними и величины Q) на практике сравнимы только по десятичным порядкам своих величин! Например, об энергетической сбалансированности судят по отличию главной значащей цифры q от 9. Нужно иметь в виду, что после этого всё еще остаётся крайне редко реализуемый, но возможный риск необычного построения исходных текстов, который правилами подсчёта V^* и E не мог быть предусмотрен априори. С этим пока пришлось столкнуться только один раз при оценивании одного профессионального фреймворка [14] (в описании модуля описания библиотечного пакета Ада встретились сотни описаний разных переменных, тогда как такие описания вообще не характерны и не рекомендованы для таких модулей в «стандартном» ООП).

Заключительные выводы и перспективы развития данного направления исследований. Выявлены источники рисков, связанных с использованием энергетических метрик и одной метрики Холстеда для выделения среди всех ПС модулей потенциально наиболее трудных. Первая группа рисков относится к неточному измерению примитивных характеристик по причинам несогласованности, отсутствия стандартных правил и проблемы точного различения таких категорий, как оператор и операнд применительно к программным символам. Показано, как сказываются ожидаемые погрешности измерения примитивов на значениях метрики объёма (позволяющей прогнозировать число ошибок кодирования), метрик трудности (по оценкам которых можно планировать распределение усилий при тестировании, доработке и модификации программ), метрик энергетической

сбалансированности (которые для модулей характеризуют стиль их разработки, а по системе в целом позволяют прогнозировать её зрелость). Наиболее значимыми рисками являются:

- несоответствие у разных стратегий подсчёта программных символов правил, определяющих учёт комментариев и омонимичных конструкций языков программирования;
- погрешность разделения словаря модуля программы на словарь операторов и операндов;
- возможность появления в профессионально разработанных программах неожиданных, с точки зрения принятых стилей программирования, фрагментов, нуждающихся в уточнении общих правил, связанных с потенциальными объёмами блоков.

За исключением последнего вида рисков, связанного с маловероятными, но непредсказуемыми событиями, для всех других рисков указаны способы предотвращения или адекватной реакции, которые при соблюдении должных условий исключают угрозы неправильных выводов и действий.

Значение данной работы состоит в том, что показана возможность системного подхода к решению проблемы превращения энергетического анализа программ из системы методов вычисления и оценки метрик в информационную технологию.

Важным прикладным результатом являются нормативы точности и рекомендации оценивания холстедовской метрики «оценки трудности», достоверность прогнозов которой сильно зависит от точности оценки её параметров. Естественным направлением развития данного исследования является сбор обширной статистики по программным продуктам и процессам для выбора обоснованных вменённых значений этих параметров.

Список литературы

1. Мищенко В. О. Математическая модель стиля Software Science для метрического анализа сложных наукоёмких программ / В. О. Мищенко // Вісник Харківського. національного університету. – 2004. – № 629, вип. 3. – С. 70–85. – (Серія : Математичне моделювання. Інформаційні технології. Автоматизовані системи управління).
2. Мищенко В. О. Энергетический анализ программного обеспечения с примерами реализации для Ада-программ / В. О. Мищенко. – Х. : ХНУ им. В. Н. Каразина, 2007. – 129 с.
3. Мищенко В. О. CASE-оценка критических программних систем. Том 1. Оценка качества / Мищенко В. О., Поморова О. В., Говорущенко Т. А. ; под ред. Харченко В. С. – Х : Нац. аэрокосмический ун-т «Харьк. авиац. ин-т», 2012. – 201 с.
4. Холстед М. Х. Начала науки о программах / М. Х. Холстед ; пер. с англ. В. М. Юфа. – М. : Финансы и статистика, 1981. – 128 с.
5. A guide to the project management body of knowledge (PMBOK® Guide). – Fourth edition. ANSI/PMI 99-001-2008. Project Management Institute, 2008. – 467 p.
6. Годунко В. М. Качество транслятора шаблонов динамических html страниц для Ada WEB серверов / В. М. Годунко, В. О. Мищенко, М. М. Резник, Д. В. Штефан // Радіоелектронні і комп'ютерні системи. – 2012. – № 5. – С. 225–229.
7. Боровинский А. В. Сложность реализации интеллектуальных графических интерфейсов для приложений, основанных на МДО / А. В. Боровинский, В. О. Мищенко // Радіоелектронні і комп'ютерні системи. – 2012. – № 7. – С. 260–265.
8. Годунко В. М. Особенности энергетических метрик UML диаграмм / В. М. Годунко, В. О. Мищенко, А. В. Пасека // Вісник Харківського національного університету. – 2013. – № 1058, вип. 21. – С. 13–19. – (Серія : Математичне моделювання. Інформаційні технології. Автоматизовані системи управління).
9. Мищенко В. О. Методы оценки энергетических метрик мобильных приложений, разрабатываемых на языке Java / В. О. Мищенко, А. Ю. Уваренко // Компьютерное моделирование в наукоемких технологиях (КМНТ-2014). – 2014. – С. 427–449.
10. Mishchenko V. O. Do the different definitions of Ada program tokens have significant difference? / V. O. Mishchenko // Радиоэлектронные и компьютерные системы. – 2008. – № 7 (34) – С. 103–106.
11. 982.2-1988 – IEEE guide for the use of IEEE standard dictionary of measures to produce reliable software. - Institute of Electrical and Electronics Engineers, 1989.
12. Mishchenko V. O. Harmonization in the formalization of a method of energy analysis

programs / V. O. Mishchenko // Electronic and computer systems. – 2008. – № 5 (32). – P. 177–182.
13. 982.1-2005 – IEEE standard dictionary of measures of the software aspects of dependability. - Institute of Electrical and Electronics Engineers, 2006. – 34 p.
14. Мищенко В. О. Метрики трудности в оценке надёжности инструментальных библиотек и фреймворков / В. О. Мищенко // Вісник Харківського національного університету. – 2014. – № 1131, вип. 25. – С. 126–147. – (Серія : Математичне моделювання. Інформаційні технології. Автоматизовані системи управління).
15. JHawk the Java metrics tool – Product Overview [Internet]. – Available from : http://www.virtualmachinery.com/ jhawkprod.htm. – Загл. с экрана.
16. Диденко Е. В. Обеспечение робастности подсчета программных символов на примере языка Ада / Е. В. Диденко, В. О. Мищенко // Компьютерное моделирование в наукоемких технологиях (КМНТ–2010) : труды науч.-техн. конф. с междунар. участием. – Х. : ХНУ им. В. Н. Каразина, 2010. – Ч. 2. – С. 84–87.

References

1. Mishchenko, V. O. (2004) Mathematical model of Software Science style for metric analysis of complex science-intensive programs. *Visnyk Harkivskogo natsionalnogo Universitety, 629 (3). Seriya "Matematychne modelyuvannya. Informatsiyni tehnologiyi. Avtomatyzovani systemy upravlinnya"*, pp. 70–85 [in Russian].
2. Mishchenko, V. O. (2007) Energy software analysis with examples of implementation for Ada programs. Kharkov: KhNU im. V. N. Karazina, 129 p. [in Russian].
3. Mishchenko, V. O., Pomorova, O. V. and Govorushchenko, T. A. (2012) CASE-assessment of critical software systems. Vol. 1. Quality assessment. Kharkov : Nats. aerokosmicheskii un-t "Khark. aviats. in-t ", 201 p. [in Russian].
4. Halstead, M. (1981) Fundamentals of programs science. Moscow: Financy i statistika, 128 p. [in Russian].
5. A guide to the project management body of knowledge (PMBOK® Guide) (2008). Fourth edition. ANSI/PMI 99-001-2008. Project Management Institute, 467 p.
6. Godunko, V. M., Mishchenko, V. O., Resnick, M. M., Shtefan, D. V. (2012) The quality of templates translator of dynamic html pages for Ada WEB servers. *Radioelektronni i komp'yuterni systemy*, (5), pp. 225–229 [in Russian].
7. Borovinsky, A. V. and Mishchenko, V. O. (2012) Complexity of the implementation of intelligent graphic interfaces for applications based on MAO. *Radioelektronni i komp'yuterni systemy*, (7), pp. 260–265 [in Russian].
8. Godunko, V. M., Mishchenko, V. O. and Paseka, A. V. (2013) Features of power metrics of UML diagrams. *Visnyk Harkivskogo natsionalnogo Universitety, 1058 (21). Seriya "Matematychne modelyuvannya. Informatsiyni tehnologiyi. Avtomatyzovani systemy upravlinnya"*, pp. 13–19 [in Russian].
9. Mishchenko, V. O. and Uvarenko, A. Yu. (2014) Methods of assessing for energy metrics of mobile applications developed in Java language. *Kompyuternoye modelirovaniye v naukoyemkih tehnologiyah" (KMNT 2014)*, pp. 427–449 [in Russian].
10. Mishchenko, V. O. (2008) Do the different definitions of Ada program tokens have significant difference? *Radioelektronnye i kompyuternye sistemy*, 7 (34), pp. 103–106.
11. 982.2-1988 – IEEE guide for the use of IEEE standard dictionary of measures to produce reliable software (1989). Institute of Electrical and Electronics Engineers.
12. Mishchenko, V. O. (2008) Harmonization in the formalization of a method of energy analysis programs. *Electronic and computer systems*, 5 (32), pp. 177–182 [in Russian].
13. 982.1-2005 – IEEE standard dictionary of measures of the software aspects of dependability (2006). Institute of Electrical and Electronics Engineers, 34 p.
14. Mishchenko, V. O. (2014) Metrics of difficulties in assessing the reliability of tools and frameworks libraries. *Visnyk Harkivskogo natsionalnogo Universitety, 1131 (25). Seriya "Matematychne modelyuvannya. Informatsiyni tehnologiyi. Avtomatyzovani systemy upravlinnya"*, pp. 126–147 [in Russian].
15. JHawk the Java metrics tool – Product Overview [Internet]. Available from:

http://www.virtual machinery.com/jhawk-prod.htm
16. Didenko, E. V. and Mishchenko, V. O.(2010) Ensuring the robustness of character counting program on the example of Ada language. *Kompyuternoye modelirovaniye v naukoyemkih tehnologiyah" (KMNT 2010)*. Kharkov: KhNU im. V. N. Karazina, (2), pp. 84–87 [in Russian].

V. O. Mishchenko, *D.Tech.Sc., professor,*
V. N. Karazin Kharkov National University,
Svobody sq., 4, Kharkov, 61022, Ukraine
mischenko@univer.kharkov.ua
S. M. Pervuninsky, *D.Tech.Sc., professor*
Cherkassy State Technological University
Shevchenko blvd, 460, Cherkassy, 18006, Ukraine
cherkpervun@rambler.ru

THE PRECISION OF ASSESSMENT OF STATIC METRICS OF SOFTWARE ENERGY ANALYSIS

Energy analysis of programs pertains to the field of SS quality control in terms of reliability. Particularly, it allows generating of judgments about difficulties of supporting separate modules of the program system and the maturity of the system as a whole. The purpose of this work is to study the risks associated with the reliability of energy analysis judgments as the result of possible inaccuracy of its metrics estimates. A number of risks is ascertained based on examination of epy sensitivity of formulas to the errors in their arguments. For this purpose statistical data on assessment of said metrics for various software systems developed in different programming languages are used. The most significant of these risks are the following ones: the inconsistency risk pertains to different strategies designed to count symbols in software as well as to rules defined for detecting comments and homonymous constructs in programming languages; the risk of objective difficulty of dividing the dictionary of the SS module develops in high level language into separate dictionaries of operators and operands; the risk of occurence in professionally designed programs of unexpected fragments (comparing with conventional programming styles and techniques), which will require making general rules, related with the module potential volumes, more precise. The last risk type becomes truth very rarely but in quite unexpected way. For all others but it, the ways are elaborated to prevent them or to respond so that the threat of erroneous judgments due to wrong values of energy metrics is excluded (if necessary conditions are met).

Keywords: *programs quality, reliability, metrics, primitive characteristics, energy programs analysis, risk, relevance, accuracy, validity.*

Рецензенти: В. М. Рудницький, д.т.н., професор,
А. І. Семенко, д.т.н., професор