УДК 004.412:004.052

Метрики трудности в оценке надёжности инструментальных библиотек и фреймворков

В. О. Мищенко

Харьковский национальный университет имени В. Н. Каразина, Украина

К надёжности инструментальных средств программирования предъявляются повышенные требования. При этом всегда актуально развитие экономичных методов оценки надёжности, применимых на всех этапах разработки библиотек и фреймворков. В данной работе развивается статический подход к оценке трудности понимания компилируемых модулей программ. Трудность или легкость такого понимания является важным объективным фактором успешности тестирования и модификации модулей. Метод статьи построен на идеях М. Холстеда, энергетическом анализе программ и приёме фильтрации модулей по трудности, связанном с известными проектами. Наш метод испытан применительно к оценке зрелости модулей системы Matreshka (из арсенала Ада индустрии), которая для контроля сравнивается с библиотекой OAE.

Ключевые слова: *надёжность программного обеспечения, статика, метрика, М. Холстед, трудность, энергетический анализ, инструментальная библиотека, фреймворк.*

До надійності інструментальних засобів програмування висуваються підвищені вимоги. При цьому завжди актуальним є розвиток економічних методів оцінки надійності, здатних до застосування на всіх етапах розробки бібліотек і фреймворків. У даній роботі розвинуто статичний підхід до оцінки труднощів розуміння модулів програм, що компілюються. Труднощі або легкість такого розуміння є важливим об'єктивним фактором успішності тестування та модифікації модулів. Метод статті побудовано на ідеях М. Холстеда, енергетичному аналізі програм і прийомі фільтрації модулів за трудністю, що пов'язаний з відомими проектами. Наш метод випробувано стосовно оцінки зрілості модулів системи Matreshka (з арсеналу Ада індустрії), яка порівнюється з бібліотекою OAE.

Ключові слова: *надійність програмного забезпечення, статичні методи, метрика, М. Холстед, тудність, енергетичний аналіз, інструментальна бібліотека, фреймворк.*

The reliability of software must meet raised requirements. Development of reliability assessment methods, which are economic and applicable at all stages of libraries and frameworks creation, stays always actual. In this paper, we develop the static approach to estimation of difficulties in understanding of compilation units. Is this unit clear or difficult to understand, that is an important objective factor affecting the success in the unit testing and modification. This article method is built on the M. Halstead's ideas, energy analysis of program and technique of units filtration by their difficulty, that is associated with well-known projects. Our method have been tested with respect to the assessment of maturity of Matreshka software units (from the Ada industry toolkit), which is compared with the OAE framework.

Key words: *software reliability, static methods, metrics, M. Halstead, difficulty, energy analysis, software toolkit, framework.*

1 Введение

Статические метрики качества программ, которые здесь обсуждаются, используют хорошо известную систему примитивных характеристик (примитивов), которую, некогда изобрёл М. Холстед. Позднее часть этой системы послужила исходным пунктом в развитии энергетического анализа

программ [1]. Первыми из числа холстедовских примитивов должны быть упомянуты словарь программного модуля η, т.е. число разных программных символов, из которых составлен исходный текст модуля, и длина программы N – общее число программных символов в модуле. Ввести определение программных символов для конкретного формального языка не сложно, но ради сравнимости оценок это целесообразно делать по аналогии примерами М. Холстеда. В [2] перечислены (и аргументированы) принципы, позволяющие это сделать. Т.н. научные метрики Холстеда [3] используют также дополнительную дифференциацию программных символов на операторы и операнды:

η_1 и N_1 – словарь и общее число операторов;

η_2 где N_2– словарь и общее число операндов ($\eta_2 = \eta - \eta_1$, $N_2 = N - N_1$).

В современных языках программирования способы различения операторов и операндов не очевидны в силу контекстных зависимостей. Более того, работа [4] показала, что такие способы по существу субъективны и, следовательно, не могут быть формализованы однозначно (например, одно и то же имя операции f при своём описании играет роль операнда, а в выражениях – оператора, и неясно, какую точку зрения предпочесть). Показательно, что несмотря на это, общий словарь и общее число программных символов практически не зависят от выбора способа различения операторов и операндов [5]. С проблемой неопределённости оценки примитивов η_1 и N_1 отчасти сталкивался и сам Холстед. Он же предложил простейший выход – для модулей, относимых к какой-то общей категории, при отсутствии точной информации полагать

$$\eta_1 : \eta_2 \approx N_1 : N_2 = \theta , \tag{1}$$

где θ - среднее по предварительно оценённой выборке или, ещё грубее, = 1/2.

В канонизированном подмножестве холстедовских метрик [3] роль операторов и операндов де факто сводится, к определению «метрики трудности»

$$\hat{D} = \frac{\eta_1}{2} \cdot \frac{N_2}{\eta_2}, \tag{2}$$

значения которой фиксируются на шкале безразмерных вещественных чисел, не меньших 1 (считается, что число η_1 разных символов-операторов в программе не может быть меньше 2, а $N_2 \geq \eta_2$ в силу определения). Точное название метрики (2) в полной системе холстедовских метрик это *оценка трудности*, поскольку Холстед дал и другой подход к характеризации трудности модуля, полагая, что «приемлема любая из двух интерпретаций» [6, С. 39]. Этот другой подход определяет *трудность D*, используя понятие потенциального объёма, введенного в [6] без точного определения. При этом подходе (2) является оценкой *D*. Большинство прикладников от использования понятия потенциального объёма отказались, и метрикой трудности *D* считают (2).

На сегодня, энергетический анализ программ строго реализует понятие потенциального объёма и определяет метрику трудности альтернативным образом по отношению к (2). Такой подход более адекватен применительно к современным программным системам, и мы его принимаем. Пусть рассматривается модуль программы или, если необходимо, его внутренний блок

(понятие, обобщающее понятия функций, процедур языков программирования [2,1]). Следуя идее Холстеда, введём потенциальный объём формулой:

$$V^* = \eta^* \log_2 \eta^* \ . \tag{3}$$

где η_2 - число формальних параметров модуля (блока). Для определения этой величины мы используем новые примитивы, допускающие точную интерпретацию в языках программирования:

$$\eta_2^* = p_1 + p_2 + j \ , \tag{4}$$

где

p_1 - число параметров шаблона или типа (если блок это шаблон или тип);

p_2 - число явных формальных параметров блока по его интерфейсу;

j - число квазиобьектов ввода-вывода в теле данного блока, полагая

$$j = \sqrt{j_1 j_2} \ , \tag{5}$$

где j_1 - число файлов, с которыми работает блок;

j_2 - число операторов ввода-вывода в эти файлы в исходном тексте тела блока.

Модули сложной внутренней архитектуры представляются [1] в виде объединения непересекающихся блоков, а их потенциальные объёмы полагаются тогда суммой потенциальных объёмов всех блоков представления.

Теперь становится корректной формула метрики трудности любого модуля

$$D = \frac{V}{V^*} \ , \tag{6}$$

где V^* потенциальный (3), а V – действительный объём данного модуля (M), который выражается через примитивы длины и словаря:

$$V = N \log_2 \eta = V(M) = V(M; N, \eta) \ . \tag{7}$$

Отметим ради истории, что в известной книге [6] фигурируют обратные к (2) и (1) величины, называемые «уровнем программы» и её «аппроксимацией».

По поводу интерпретации наблюдённых значений $\hat{D}$ документ [3, С.84] лаконично констатирует, что «…для средней программы PL/I, трудность должна быть не менее 115. Трудность больше, чем 160, указывает на серьезные проблемы. Данные о трудности доступны также и для других языков». Учитывая беглые замечания некоторых других авторов, публиковавшихся на рубеже 70-80 гг., эту информацию можно понимать как следующую рекомендацию: На основе статистических данных априори разделять модули большой программной системы с точки зрения риска ошибочной реализации спецификаций на 3 класса (один из которых содержит модули, не внушающие подозрений). При этом использовать 2 рубежа трудности:

$$D_1 = E\left(\hat{D}(M_1), .. \hat{D}(M_N)\right), \qquad D_2 = E + S \qquad , \tag{8}$$

где E – среднее значений трудности (2) по базовой выборке модулей M_i, которые разрабатывались в данной системе программирования на данном языке;

S – стандартное отклонение в той же выборке.

Более точная информация содержится в малодоступных технических отчётах фирмы IBM. Однако основные моменты находим в фундаментальном обзоре [7], фактически подбившем итог эпохи интенсивных исследований по направлениям холстедовской науки о программах. Из наиболее важного для нас, процитируем следующее место [7, С.160], относящееся к отчёту С. П. Смита из лаборатории указанной фирмы в Санта Терезе: «… последнее исследование предложило учредить пороговые значения $\hat{D}$, которые могут быть использованы программистами в качестве руководства для разработки программных продуктов. Например, для программ PL/S, было установлено, что среднее значение η_1 должно быть 46, а отношение N_2/η_2 – меньше 5 ... Два пороговых значения для сложности метрики могут быть определены следующим образом:

$$\hat{D}_1 = (46/2) \times 5 = 115 \ , \qquad (22)$$

$$\hat{D}_2 = (46 + 18/2) \times 5 = 160 \ . \qquad (23)$$

(Значение 18, использованное в (23), является стандартным отклонением η_1) Если для определенного модуля $115 < \hat{D} < 160$, то программисту рекомендуется пересмотреть свой код с точки зрения наличия неудачных приёмов программирования, таких как использование слишком большого числа операторов GO TO, неоправданного обилия встроенного кода на ассемблере или необоснованного использования дополнительных операндов. Если $\hat{D} > 160$, то рекомендуются более радикальные меры, такие как экспертиза в группе. Исследования IBM позволяют заключить, что подобные пороговые значения могут быть установлены для других языков программирования высокого уровня».

Подводя итог, подчеркнем вывод о том, что метрику оценки трудности можно, основываясь на опытных данных, использовать для предварительной классификации модулей программной системы по степени их понятности для исполнителей при тестировании, устранении дефектов или модификации системы. При этом в силу психофизической подоплёки дела является очевидным, что определение соответствующих порогов с высокой точностью, даже, если они хорошо выражены, не критично для успешности метода. Представляется, что опубликованные значения порогов для конкретной системы программирования могли бы служить ориентиром и непосредственно использоваться до накопления собственного опыта.

Те статические методы оценки качества программ, которые требуют использования алгоритмов только линейной или близкой к ней сложности (таковы, например, алгоритмы оценки метрик трудности!), всегда привлекательны в силу малости затрат при использовании. В частности, задачи оперативного сравнения больших систем, например, новых инструментальных или прикладных библиотек, с общих позиций сложности, рисков остаточных ошибок и т.п. могут решаться с помощью научных или энергетических метрик,

что подтверждается систематическим появлением публикаций на такие темы (из недавних [8-11]). Однако техника фильтрации, основанная на (2), с эпохи 80-х годов не обновлялась, а возможность использования в этом приёме метрики (6) вместо (2) является чистой гипотезой (хотя и правдоподобной).

Актуальность развития в современных условиях методов использования метрик трудности раздельно компилируемых модулей программ наглядно подтверждается следующим фактом. При модернизации документа [3] холстедовские метрики были в [12] исключены из перечня рекомендованных для стандартного использования инструментов контроля надёжности критических систем со следующим объяснением причин: «n_1, n_2, N_1, N_2 оказались полезными в Спейс-Шаттл в роли дискриминантов качества. Тем не менее, эти метрики трудно внедрить, потому что пороги, которые отличают низкое качество от высокого, должны быть оценены статистически. Этот процесс выходит за пределы навыков большинства практиков» [12, С. 24] (в цитируемом документе n_1, n_2 означают то, что выше обозначалось как η_1, η_2).

Таким образом, не ставя под сомнение полезность метрики $\hat{D}$, основанной на примитивах η_1, η_2, N_1, N_2 , авторы документа констатировали невозможность их механического использования сегодня с помощью прежней техники. Необходимы новые исследования для уточнения и восстановления доверия к данной метрике. Что же касается метрики D (6), то она в прикладном аспекте не исследована и, если упоминается, то в обзорном порядке.

Целью работы является создание и обоснование решающих правил, необходимых для реализации идеи фильтрации модулей программ на основе степени трудности, с испытанием на примере реальной программной системы .

2 Постановка задач данного исследования

Возможность применения метрики оценки трудности (2) к современным проектам мы рассмотрим на примере сравнения степени зрелости инструментальных библиотек Ада программирования [13,14]. При этом принимается во внимание, что уже в проектах с числом модулей порядка 10^3, и, тем более, в больших проектах, ни менеджеры качества, ни группа сопровождения обычно не располагают возможностью тщательного просмотра всего программного кода. Выборочные методы контроля не снимают этой экономической проблемы полностью, поскольку из соображений репрезентативности, выборки должны насчитывать, порядка 10^2 модулей и больше. Только после удачной классификации модулей можно сосредоточиться на тех из них, которые наиболее уязвимы с точки зрения рисков дефектности кода, трудности исправления выявленных ошибок или неудобства модификаций. Нашей первой задачей является *разработать современный вариант фильтрации модулей по классической метрике оценки трудности кода*. При этом в своих экспериментах с кодом, написанным на языке Ада, мы примем огрубляющее предположение (1), по-видимому, неизбежное для языков достаточно высокого уровня. Оно не лишает оценку (2) содержательности, а там, где возможны «точные» правила различения операторов и операндов выводы из решающих правил слабо чувствительны к таким огрублениям, как отмечал еще Холстед [6].

Вторая задача могла бы состоять в том, чтобы предложить аналогичную классификацию с использованием метрики трудности (6). Однако в результате корректного определения потенциального объёма, которое удалось достичь в энергетическом анализе [1,2], отношение (6) теряет однородность. В его числителе осталась бы величина лексической природы, а в знаменателе теперь появился бы числовой атрибут характеристики, отражающей наличие внутренних и внешних интерфейсов модуля. Кстати говоря, там, где его можно в духе Холстеда как-то оценить, потенциальный объём занижается, так как не учитывает внутренней архитектуры и всех возможных параметров. Поэтому в энергетическом анализе было естественным (в частности, с точки зрения противодействия систематическому смещению оценок в сравнении с наукой о программах) определять трудность на основе более адекватной (кстати, всегда не меньшей объёма(7)) величины *объёма разработки* модуля, полагая

$$D = W / V^* \qquad (\textit{трудность модуля} \text{ программы}) \tag{9}$$

где объём разработки W прибавляет к (7) вклады отношений между модулями (отношения статические, и, отчасти хронологические):

$$W(M) = V(M; N, \eta') + \sum_i V(M_i) \tag{10}$$

где M_i –интерфейсные модули системы, от которых зависит данный модуль, и которые были созданы раньше его;

V – холстедовский объем модуля (7);

η' – полный словарь нашего модуля M ($\eta' \geq \eta$), определяемый так, чтобы учитывать контекст этого модуля в программной системе (подробнее в [1-2]). В частном случае изолированного модуля получается $W = V$, и D (9) совпадает с трудностью в смысле Холстеда [6].

В итоге, ставится задача *разработки метода классификации модулей по обновлённой метрике трудности из энергетического анализа программ* (9).

Третья частная задача исследования - это *уточнение вопроса о связи между метриками трудности и оценки трудности*. М. Холстед в [6] обосновывал точку зрения о том, что (2) и (6) могут быть на практике отождествлены, но в дальнейшем исследователи пришли к другому выводу: это существенно разные метрики [15]. В работах [9-11] высказаны гипотезы о том, что, будучи разными, метрика (2) и метрика (9), обобщающая (6), могут быть взаимозаменяемы в функции фильтрации модулей по их трудности. Это было бы неплохо потому, что нередко одну из них технически трудно использовать, а другая доступна.

Наконец, в прикладном плане мы попытались на основе формального анализа с помощью подходящей выборки охарактеризовать трудность модулей системы фреймворков Matreshka [13]. Это не малая инструментальная система программных модулей, предназначенных для профессионального использования в проектах на языке Ада, которая быстро развивается. Её можно сравнивать с априори более зрелой разработкой инструментальной библиотеки OEM [14], которая, продолжая развиваться, в значительной части утилизирует прошедшие испытание временем разработки 1999-2004 гг из проекта Gnatcom.

3 Материалы исследования, относящиеся к метрике оценки трудности

По сути, необходимо проверить, что классификация модулей в соответствии с величиной оценки трудности позволяет оставить в одном полярном классе, соизмеримом по величине со всей выборкой, «лёгкие» модули, а в другом, меньшем (скажем, на порядок), чем выборка, трудные модули. Оговорим, что часть Ада модулей проекта [13] генерировалась автоматически, для них понятие трудности лишено обычного смысла, и они не рассматривались.

Из модулей разработанных программистами-людьми была с участием В. М. Годунко сделана репрезентативная выборка объёма 74. Проведена оценка научных и энергетических метрик с использованием программного анализатора SS_Ada_Scanner [2], который разрабатывался для стандарта Ада-95. Часть этой работы выполнила А. И. Паньковская при прохождении магистратуры в соответствии с планом дипломного проектирования. Поскольку система Matreshka создавалась на языке стандарта Ада-2005 с переходом на Ада-2012, то изредка новые языковые конструкции воспринимались анализатором в отдельных модулях как синтаксические ошибки с плохой локализацией места «ошибки». Это затруднение преодолевалось нами вручную – путем локальной «микромодификации» исходного текста без изменения результата оценки или с его искажением на какие-то доли процента.

Переходя к классификации, будем исходить из полезности порогов трудности (8), наличие которых отражает ограниченность комбинаторных возможностей ума. Как мы отмечали, они были подсчитаны на примере PL/S программирования, но определялись, по сути, психофизическими нормами, а потому нельзя ожидать их кардинального изменения с изменением языка программирования.

Очевидно, однако, что значения $\hat{D}$ в современных программных системах будут варьироваться намного заметнее с учётом разнообразия технологий программирования и возросшей автоматизации этого процесса в компьютерных средах разработки. Поэтому имеет смысл сразу ввести дополнительные классы трудности модулей, воспользовавшись изложенным в [7] опытом. Действительно, применим известное эвристическое положение В. И. Романовского о практической универсальности «правила трёх сигм». Тогда для большинства модулей M той выборки, с помощью которой были образованы «стандартные» пороги, должно выполняться неравенство:

$$\eta_1(M) \le E(\eta_1) + 3\sigma(\eta_1)\,, \tag{11}$$

где E – среднее по выборке, а σ – выборочное стандартное отклонение.

Следовательно, имеет смысл нанести на шкалу метрики оценки трудности дополнительный ориентир:

$$\hat{D}_3 = 0.5 \cdot (E(\eta_1) + 3\sigma(\eta_1)) \cdot \min N_2/\eta_2) = 0.5(46 + 3 \cdot 18) \cdot 5 = 250\ . \tag{12}$$

Имеет смысл ввести также аналогичные пороги

$$\hat{D}_4 = 0.5 \cdot (E(\eta_1) + 5\sigma(\eta_1)) \cdot \min N_2/\eta_2) = 340\ , \tag{13}$$

$$\hat{D}_5 = 0.5 \cdot (E(\eta_1) + 7\sigma(\eta_1)) \cdot \min N_2/\eta_2) = 430\ . \tag{14}$$

Если за пятым порогом $\hat{D}_5$ окажется число модулей, явно превосходящее численность соседних добавочных классов, то, очевидно, либо классификация оказалась узка, не современна, либо при разработке кода изучаемой системы не стремились соблюсти необходимое условие понятности модулей – их обозримость. При этом полярные классы (с наименьшими и наибольшими значениями $\hat{D}$) будут играть ту же роль, что 1-й и 3-й в классификации [6], а промежуточные классы распределят между собой роль среднего класса стандартной классификации.

Сказанное послужило основанием для следующих определений, где, однако, интерпретация трудности в форме требований к самоконтролю и контролю дана в качестве примера, и в зависимости от условий разработки и степени её критичности может конкретизироваться или корректироваться.

Определение 1. Классы трудности по метрике $\hat{D}$ определяются указанными выше пороговыми значениями:

0 класс, $\hat{D} \le D_1$ – модули, не требующие особого внимания, кроме обычного контроля разработчика («лёгкие»);

1 класс, $D_1 < \hat{D} \le D_2$ – модули, требующие дополнительного тестирования со стороны самого разработчика («умеренно трудные»);

2 класс, $D_2 < \hat{D} \le D_3$ – модули, требующие повышенного внимания со стороны разработчика, дополнительного тестирования другим исполнителем («довольно трудные»);

3 класс, $D_3 < \hat{D} \le D_4$ – модули, требующие со стороны разработчика повышенного внимания, тестирования другими исполнителями с сохранением отчётов о тестировании, возможен пересмотр кода для его упрощения («трудные»);

4 класс, $D_4 < \hat{D} \le D_5$ – модули, требующие выяснения причин высокого показателя трудности, создания предварительного плана тестирования, обсуждения в группе кода и/или тестов и/или результатов тестирования, показан пересмотр кода в смысле улучшения стиля («очень трудные»);

5 класс, $D_5 < \hat{D}$ – модули, требующие особого внимания, выяснения причин ненормально высокого показателя трудности, обязательного улучшения кода всюду, где нарушен хороший стиль, и передачи его на экспертизу в другую группу или/и декомпозиции (модификация) в соответствии со смыслом с целью кардинального снижения трудности («ненормально трудные»).

С неформальной точки зрения, в зрелой системе, как минимум, половина компонент не должны быть трудными для понимания. Численность класса максимальной трудности не должна превышать примерно 10%. Формализуем эти требования с разумной поправкой на число предусмотренных классов (у нас их всегда будет 6, но [6] рассматривалось 3, и можно представить другие числа в других классификациях).

Решающее ***правило*** **1**. Пусть имеется метрика модулей программ, которая имеет интерпретацию трудности понимания исходного кода и шкалу для классификации модулей по значениям этой метрики, определяя K классов, $3 \le K \le 9$. Программную систему будем считать компонентно зрелой по данной

метрике тогда, когда при $K \geq 5$ не менее половины всех модулей программы лежат в классе наименьшей трудности (0-м классе), а при $K<5$ там находится не меньше $\{(2K\text{-}3)/(4K\text{-}8)\}\cdot 100\%$, и в любом случае в классе максимальной трудности содержится не более $3/2(K+9)$ модулей.

При $K = 6$, класс максимальной трудности по этому определению не должен включать более десятой части всех рассмотренных модулей.

Это правило всё еще не вполне формально по двум причинам. Во-первых, системы могут быть компонентно неоднородными (одни файлы содержат исходные тексты на основном языке программирования, другие – ресурсные и т.п.). Во-вторых, редко обследуется система целиком, чаще – выборка модулей. Если она образована с использованием случайного выбора, то численности классов трудности – случайные величины, и требуется уточнить способ их сравнения с детерминированными значениями. Однако обе эти причины, как правило, связаны с уточнениями «на втором-третьем знаках после запятой». С другой стороны, если их принимать во внимание, то нужно знать конкретные обстоятельства обследования, в общем случае непредсказуемые. Ограничимся одним общим уточнением: контролировать репрезентативность используемых выборок. В частности, как принято в практической статистике, желательно, чтобы объёмы выборок были не меньше 50.

Табл. 1 показывает предварительное распределение 74 модулей обследуемой системы по значениям метрики оценки трудности. Оно имеет «длинный хвост».

Табл. 1 Распределение оценок трудности модулей в исследуемой выборке из Matreshka

$\hat{D} \leq 10^2$	$10^2 \leq \hat{D} \leq 2\cdot 10^2$	$200 \leq \hat{D} \leq 4\cdot 10^2$	$400 \leq \hat{D} \leq 8\cdot 10^2$	$800 \leq \hat{D} \leq 16\cdot 10^2$	$1600 \leq \hat{D} \leq 32\cdot 10^2$	$3200 \leq \hat{D} \leq 64\cdot 10^2$	$6400 \leq \hat{D} \leq 128\cdot 10^2$	$128\cdot 10^2 \leq \hat{D}$
$n =$ 36	$n =$ 11	$n =$ 6	$n =$ 5	$n =$ 9	$n =$ 1	$n =$ 2	$n =$ 0	$n =$ 4

По классам рассмотренные модули распределились так, как показано в табл. 2. Распределение 70% модулей по классам 0-4 демонстрирует выраженную компонентную зрелость. Однако попадание оставшихся 27% в класс ненормальной трудности указывает на вероятность того, что некий пласт модулей в исследованной версии системы формальному критерию компонентной зрелости не удовлетворяет. К выяснению причин мы вернёмся.

Табл. 2 Частоты и частости попадания модулей Matreshka в классы трудности по $\hat{D}$

Класс	0	1	2	3	4	5
Частота	39	4	6	2	2	21
Частость	0.527	0.054	0.081	0.027	0.027	0.284

4 Данные по системе, используемой для сравнения

Многие нужды пользователей, которые обеспечиваются мультиплатформенной системой Matreshka, могут быть под Windows обеспечены также библиотекой OEM [14], в большей степени ориентированной на бизнес-приложения. История её разработки и более широкий круг пользователей позволяют неформально характеризовать её как зрелую

программную систему, и ожидать выполнения критерия компонентной зрелости. При этом нужно иметь в виду асимметрию этого сравнения по смыслу. Поскольку изучаемой проблемой является степень зрелости системы Matreshka, то выборка её модулей формировалась с использованием рандомизации ради обеспечения репрезентативности. Однако для сравнительной оценки на основе научных и энергетических метрик желательна сопоставимость выборок. Для этого выборку Matreshka разделили на 3 части – большие, средние и малые по объёму модули с тем, чтобы такие модули в той же примерно пропорции были представлены в контрольной выборке из OEM. Выбор их был случаен, но после исчерпания квоты малых модулей, такие в дальнейшем не включались. По выполнении квоты для средних, пропускались и они, пока не набралось нужное число больших модулей. Разделять на большее число частей нельзя, поскольку большие модули из OEM в среднем намного меньше, чем большие из Matreshka.

Поскольку объём и оценка трудности между собой часто (положительно) коррелируют, то в случае обнаружения признаков одинаковой компонентной зрелости, напрашивающийся вывод имел бы, в силу способа формирования выборки, сомнительную значимость. Однако при обнаружении признаков различия, напротив, вывод о различии можно в данном случае делать с большей уверенностью, чем, если бы выборка из OEM была совершенно случайной.

Из 326 модулей ядра OEM была описанным способом образована выборка из 62 модулей (не взяли 74, поскольку не хватило крупных модулей). Их распределение по метрике оценки трудности показывает табл. 3. В отличие от Matreshka, где наблюдался сильно вытянутый хвост, модули из OEM ограничены рубежом, примерно равным 3000, причём в 87% случаев не переступают и рубеж 400.

Табл. 3 Распределение оценок трудности модулей в контрольной выборке из OEM

$\hat{D} \le 10^2$	$10^2 \le \hat{D} \le 2 \cdot 10^2$	$200 \le \hat{D} \le 4 \cdot 10^2$	$400 \le \hat{D} \le 8 \cdot 10^2$	$800 \le \hat{D} \le 16 \cdot 10^2$	$1600 \le \hat{D} \le 32 \cdot 10^2$	$3200 \le \hat{D} \le 64 \cdot 10^2$	$6400 \le \hat{D} \le 128 \cdot 10^2$	$128 \cdot 10^2 \le \hat{D}$
$n =$ 33	$n =$ 10	$n =$ 11	$n =$ 5	$n =$ 2	$n =$ 1	$n =$ 0	$n =$ 0	$n =$ 0

Распределение по классам трудности представлено в табл. 4. Модули OEM в 53% случаев лёгкие, и менее чем в 10% случаев попадают в класс ненормально трудных модулей. Критерий компонентной зрелости относительно $\hat{D}$ в OEM можно считать выполненным. На самом деле он выполнен ещё более рельефно, ведь по способу образования контрольной выборки, в неё вошли все наиболее крупные модули, а меньшие и мелкие только в 1/5 от их числа в системе.

Табл. 4 Частоты и частости попадания OEM модулей в классы трудности по $\hat{D}$

Класс	0	1	2	3	4	5
Частота	33	7	8	5	3	6
Частость	0.532	0.113	0.129	0.081	0.048	0.097

5 Новый метод, относящийся к метрике трудности

Замещение теоретически обоснованной метрики трудности метрикой оценки трудности (которую часто именуют просто «метрикой трудности», как в [3]) предполагало их приближенное равенство. Однако в некоторых случаях значения трудности D на порядки отличаются от значений оценки трудности $\hat{D}$.

При этом метрика D обычно даёт больший разброс по модулям большого проекта, чем $\hat{D}$. При моделировании её значений случайными величинами рискованно предполагать, что они имеют конечные средние (математические ожидания)! Поэтому нормативы, рассчитанные для $\hat{D}$ на основе средних, следует использовать при анализе распределения D на модулях проекта с крайней осторожностью (например, в только начальном приближении или при свидетельствах усиленной корреляции), как в [9].

Мы преодолеем указанную трудность, используя разделение на классы по самой выборке. Это можно делать, поскольку, в отличие от $\hat{D}$, трудность D никак не связывается с психофизическими нормативами. Она безразмерна и является, как в (7), так и в (9), грубо говоря, коэффициентом усложнения исходного текста, рассматриваемого как реализация спецификаций на данном языке программирования в сравнении с воображаемым эталонным представлением тех же спецификаций. Сравнимость классов трудности, выделенных в разных проектах, будет обеспечиваться их образованием на основе квантилей наблюдённых распределений. Введём 6 классов трудности для анализа на основе метрики D, имея в виду попытаться их сопоставить классам трудности, основанным на $\hat{D}$. Рекомендации действий снова условны.

Определение 2.

0* класс образуют все те модули (системы или исследуемой выборки), для которых трудность D не превосходит медианы значений этой метрики на рассматриваемых модулях; предполагается, что такие модули не требуют иного внимания, кроме обычного контроля разработчика;

классы с 1* по 4* определяются как соответствующие децили (с 6-го по 9-й) распределения трудности D на рассматриваемых модулях; при этом 1*-й класс содержит модули, также не требующие, по предположению, особого внимания, если на противное не укажет другой используемый индикатор трудности; модули 2*-го класса, предположительно, нуждаются в дополнительном тестировании со стороны самого разработчика, если только другой используемый индикатор трудности не позволит от этого отказаться; модули 3*-го класса рассматриваются, как безусловно требующие дополнительного интенсивного тестирования со стороны разработчика; в 4*-м классе находятся модули, требующие повышенного внимания и дополнительного тестирования другим исполнителем (или обсуждения в группе);

5* класс образуется как 10-й дециль распределения трудности D на рассматриваемых модулях; его представители требуют особого внимания, выяснения причин ненормально высокого показателя трудности (в частности, в сравнении с индикацией по другим критериям), показана их передача на

экспертизу в другую группу или декомпозиция (модификация) с целью кардинального снижения трудности.

Для определённости, если число модулей $m = 10k+2l+s$, $0 \le l \le 4$, $0 \le s \le 1$, то 1* класс полагаем содержащим $5k+l+s$ модулей, 5* класс – k, а из остальных первые l классов – по $k+1$ модулю, оставшиеся (если будут) – по k модулей.

Замечание 1. Особенность определения 2 состоит в том, что по принятому выше решающему правилу 1 всякая система модулей, отфильтрованных в классы трудности D, будет «считаться» компонентно зрелой по метрике трудности D. Дело в том, что данная классификация не нацелена на проверку зрелости систем в силу отсутствия у метрики D собственных абсолютных ориентиров приемлемой и/или неприемлемой трудности. Однако определение 2 согласуется с правилом 1 в предположении, что классификация будет применяться, как правило, к зрелым системам.

Вернёмся к рассмотренной в разделе 4 выборке модулей из безусловно зрелой системы OEM. Учитывая, что $m = 62 = 10 \cdot 6+2 \cdot 1$, и, отыскав децили, мы можем показать рубежи между классами (беря полу-суммы граничных элементов соседних классов или граничный элемент более избыточного класса):

$$D_1^* = 76{,}5\ , \quad D_2^* = 120{,}3 \quad D_3^* = 183{,}3 \quad D_4^* = 409{,}7\ , \quad D_5^* = 1152{,}0\ . \quad (15)$$

Позволят ли наблюдения за большим числом различных проектов подобрать на основе таких рубежей собственные пороги для метрики трудности D, сомнительно в силу сказанного выше. Однако, такие рубежи дают очевидный способ сравнения разных проектов по трудности их модулей.

Проведём такое сравнение с помощью уже использованной в разделе 4 выборки 74 модулей из системы Matreshka, которую мы исследуем на наличие разных признаков зрелости. Для неё получились следующие межклассовые рубежи:

$$D_1^* = 125{,}5, \quad D_2^* = 283{,}9 \quad D_3^* = 389{,}0 \quad D_4^* = 1068{,}0\ , \quad D_5^* = 5472{,}6\ . \quad (16)$$

Сравнение (16) и (15) показывает, что модули Matreshka имеют существенно большую трудность в смысле метрики D, чем модули OEM. А именно, налицо «опережение на класс»: первый рубеж трудности Matreshka (D_1^*) больше, чем второй (D_2^*) для OEM, второй – больше 3-го, третий примерно равен 4-му (точность – 0,5%), а четвёртый – 5-му (1,5%). Только один, самый трудный модуль из OEM выборки, имеет значение трудности (6461,1), соответствующее наивышему классу трудности Matreshka (который насчитывает 7 модулей).

Несмотря на то, что метрика энергетическог анализа D позволяет сравнивать трудность разных систем, но не зрелость, зрелость систем и даже отдельных модулей в энергетическом анализе также оценивается. К этому и перейдём.

6 Сравнение с результатом применения энергетического критерия

Оставаясь в рамках сравнения статических характеристик, к вопросу зрелости программного кода можно подойти иначе, чем в разделе 3. Согласно

математической модели энергетического анализа программ [1,2] спецификационная энергия и работа программирования связаны между собой законом сохранения, который учитывает также ментальные энергетические затраты разработчика:

$$E = A + Q\,, \tag{17}$$

где E – спецификационная энергия рассматриваемой подсистемы S;

A – работа, затраченная в процессе разработки программного кода S;

Q – суммарное интеллектуальное тепло, полученное от разработчика в этом процессе.

В концептуальном плане данная модель и её уравнение (17) аналогичны хорошо известной модели термодинамики и уравнению её первого начала. Подобно тому, как это имеет место при изучении термодинамических процессов, уравнение сохранения энергии в анализе программ делает возможной количественную оценку эффектов, которые непосредственно не измеряются (в физике это движения молекулярного уровня, а в программировании – творческая составляющая работы человека над кодом). При этом соотношение между достигаемым в процессе разработки значением спецификационной энергии смыслового модуля программы и затраченной на это работой,

$$A = k \cdot E\ , \tag{18}$$

определяется типом процесса [1,2], а он, в свою очередь, – использованной технологией программирования (а её выбор может зависеть от типа модуля). Ход процесса обычно недоступен для измерений, и тогда используется итоговая характеристика процесса, причём обычно вместо Q удобно рассматривать безразмерную величину нормированного информационного тепла [1,2]:

$$\mathrm{q = Q/max(E,A)} \tag{19}$$

где работа программирования отдельного модуля определяется с помощью характеристик (5), (7) формулой:

$$A = {W^2}/{V^*}\ , \tag{20}$$

а спецификационная энергия интерфейсного модуля, если он составлен из некоторого числа не сгруппированных блоков, имеет оценку через энергию его свободной группы (в исследованных модулях других групп с заметной энергией не встречалось):

$$E_{free} = \begin{cases} \lambda^{-2}(V^*)^3 & \Leftarrow m < 5 \\ \lambda^{-2}\dfrac{(V^*)^3}{m/9 + 0.5} & \Leftarrow m \geq 5 \end{cases}, \tag{21}$$

где m – количество блоков в модуле,

λ– уровень языка программирования (в наших расчетах λ=1.66).

Вынужденно опустим здесь имеющие отношение к (18) - (21) многочисленные и важные варианты, подробности, обоснования, предположения, упрощения и оговорки, которые можно найти в [3,1]. Ограничимся изложением смысла и техники проведенных оценок. Системы, разработанные на языке Ада, физически состоят из интерфейсных модулей в форме спецификаций библиотечных пакетов, тел таких пакетов (при этом не все спецификации нуждаются в телах), а также библиотечных подпрограмм, субмодулей и др. Однако рассматриваемые в статье системы составлены, в основном, из пар спецификация-тело, которые определяют

$$E_i = E(M_i), \quad A_i = A(M_i) + A(bM_i), \quad Q_i = E_i - A_i, \tag{22}$$

где M_i - описание i-го библиотечного пакета (самостоятельный компилируемый модуль, который, как правило, находится в отдельном файле);

bM_i - тело i-го библиотечного пакета (самостоятельный компилируемый модуль, который, как правило, находится в отдельном файле).

Для программной системы в целом или её подсистемы (в т.ч. репрезентативной выборки модулей) рассматриваемые метрики определяются суммированием:

$$E = \sum_i E_i, \quad A = \sum_i A_i, \quad Q = \sum_i Q_i = E - A. \tag{23}$$

Необходимо подчеркнуть, что величина A в (17), будучи прямым обобщением холстедовского усилия, может трактоваться как нормализованное время программирования [6,3], тогда как соответствующая величина (20) – это метод вычисления метрики программного кода. Связь между ними состоит в том, что метрика характеризует затраты на создание кода в неких идеализированных стандартных условиях (информация, необходимая для оценки реальных затрат [2], обычно отсутствует). В связи со сказанным, будем дальше трактовать (18), как соотношение между метриками кода. Теоретические соображения и многочисленные наблюдения за исходными кодами (причём на разных, вообще говоря, языках программирования, в частности, Ада) свидетельствуют о наличии тенденции к тому [2,1], что в зрелой программе величина k по порядку величины равна 1, а, значит, Q мало в сравнении с абсолютными величинами E и A, а, значит, q просто мало. Шкала для q уточняет это обстоятельство, а также отражает ту реальность, что необходимые оценки, в особенности, уровня языка программирования, нередко имеют погрешность на уровне 10%, так что оценки при k = 1 и, скажем, при k = 1.25 или 0.8, как правило, не имеют принципиального различия. Добавим, что значению k = 10 соответствует q = - 0.9, k = 2 соответствует q = - 0.5, k = 0.5 - q =+0.5, а k = 0.1 - q =+0.9. Если $k \to \infty$, то $q \to$ - 1, а если $k \to 0$, то $q \to +1$.

Решающее ***правило 2***. Программную систему или подсистему (в частности, отдельный модуль или логический модуль, объединяющий интерфейсный модуль и его тело вместе, если есть, со всеми субмодулями тела) считать *энергетически сбалансированной*, если

$$-0.9 \le q \le 0.9 \tag{24}$$

и *хорошо* (*энергетически*) *сбалансированной*, если

$$- 0.5 \le q \le 0.5 \qquad (25)$$

Соответственно подсистема (энергетически) *не сбалансирована* при $|q|>0.9$.

В силу сказанного выше, для завершенных в разработке, эксплуатируемых или проходящих испытания систем энергетическая сбалансированность интерпретируется как показатель зрелости этой системы. Такой признак зрелости назовём *энергетической зрелостью*. При этом сбалансированность более показательна для системы в целом или её представительной части, чем для отдельного модуля и малых подсистем. Для них знак и величина q, очевидно, зависят от специализации модулей в системе, так что даже в хорошо сбалансированной системе процент сбалансированных логических модулей может быть довольно мал. Этот процент представляет статистический интерес при анализе, но в суждении о зрелости системы его роль второстепенна.

Применим правило 2 к рассматриваемым программным системам. Результаты отражены в табл. 5. Несбалансированность для системы модулей Matreshka, так же как сбалансированность для OEM хорошо различаются по численным значениям метрики нормированного информационного тепла. Процент сбалансированных логических модулей в OEM также выше, но сам по себе не высок, что, как пояснялось, не является значимой особенностью.

Табл. 5 Показатели энергетической сбалансированности в системах Matreshka и OEM

Сбалансированность модулей :	выборка в целом	хорошо сбалансированных	всех сбалансированных	несбалансированных
Matreshka (43 лог.модулей)	нет, q= -0.9999995	2.3% (1)	20.9% (9)	79.1% (33)
OEM (31 лог.молуль)	есть q = 0.81	3.2% (1)	32.3% (10)	65.0% (20)

7 Анализ результатов и выводы

Формула (2) для $\hat{D}$ вполне содержательна, и первоначально не возникает подозрений по поводу приведенной нами цитаты (С. 130) из документа [12] относительно трудности оценки пороговых значений для данной метрики. Можно, однако, рассуждать так. В грубом приближении оценка $\hat{D}$ (2) примерно пропорциональна длине программы, а, точнее, близка к значению $N/4$. В упрощающем предположении (1), которым широко пользовался сам Холстед, это очевидно. Но и в общем случае, по нашим наблюдениям, «более точная» оценки редко отклоняется от четверти длины больше, чем на 25-30%. Подоплёка в том, что символ-оператор, как правило, не выступает без операндов, а операнды не накапливаются ни в каком месте программы без разделителей, т.е. операторов. В итоге, попадание модулей в классы трудности зависит в главном от размеров модулей. Не прозрачен ли в той же степени вопрос о порогах между классами, использованными в разделе 3?

Во все времена эпохи массового программирования при разработке исходных кодов используются персональные терминалы и распечатка текстов на бумагу. Наиболее надёжно человек анализирует то, что может охватить одним взглядом, т.е. то, что умещается на странице (или в одном-двух окнах на экране монитора). В разные времена и при разных форматах это могло составлять примерно от 16 до 60 строк (например, в тексте этой статьи страница вмещает до 45 строк, а в GPS [17] - популярной среде подготовки Ада программ – окно редактора позволяет обычно видеть до 30 строк одновременно). Положим по максимуму - 60. Многолетний опыт энергетического анализа показывает, то в исходных текстах программ на ЯВУ (за вычетом из них пустых строк и многострочных комментариев) среднее число программных символов на одной строке редко выходит за диапазон 3 - 10, а чаще колеблется на уровне 5 - 7 (например, для Matreshka этот показатель составляет 5.04±1.13 при медиане 5.16). Берём 7. Получим оценку максимального размера текста программы, удобного для просмотра, как $N/4 \approx 60 \cdot 7/4 = 105.0$, т.е., близко к $D_1=115$ [3,7]. Если взять предельную оценку числа программных символов на строке, то получим $60 \cdot 10/4 = 150.0$, т.е., близко к $D_2=160$ [3,7]. Следовательно, *проводя анализ иначе*, чем это делали авторы, предложившие пороги трудности для исходных текстов программ лет 30-35 тому назад, и, исходя из других, но тоже вполне достоверных (без претензий на высокую точность) оценок психофизичеких возможностей человека, *мы приходим к аналогичным оценкам этих порогов*!

Стоит ли тогда подвергать ревизии эти пороги в современной программометрии? Приведенная выше фраза [12] о том, что «пороги, которые отличают низкое качество от высокого, должны быть оценены статистически» не вызывает возражений в принципе, но она очевидно, отражает негативный опыт попыток извлечь уточнённые значения порогов трудности из современных программ прежними методами. Это в действительности невозможно! Табл. 1 хорошо иллюстрирует ситуацию с современным программированием, когда наряду с тенденцией к ограничению размеров отдельно компилируемых модулей ради упрощения процесса (около 63.5% модулей имеют оценку <200) в современных средах программирования создаются также и сколь угодно большие модули (около 16% модулей нашего примера имеют оценку трудности около $1\text{-}5 \cdot 10^3$, а 4 – оценку >12800). В таком случае среднее (у нас 4559) и стандарт (25001) оказываются слишком велики для того, чтобы без большого риска служить ориентирами (в нашем примере 94.6% модулей попали бы в 0-й класс легких, и, не потребовали бы к себе внимания, хотя многие имеют $\hat{D}$, раз в 10 превосходящее классические пороги!). Ясно, что в других системах такие оценки давали бы совсем иной результат (например, для OEM первый подобный «порог» в 22 раза, а верхний в 51 раз меньше!). *Правильный выход из положения состоит в увеличении числа классов трудности с сохранением неизменного порога для лёгких модулей*. Это решение имеет прозрачный смысл. Почему сегодня можно успешно работать с текстами значительных размеров? Благодаря удобной навигации и гипертекстовому подходу языковых процессоров. Например, если неясен смысл элемента программы, то мы одним нажатием курсора открываем дополнительное окно с тем участком программы (или

документации), где этот элемент определён. Если модуль занимает несколько страниц, такие переходы не очень-то напрягают программиста, но при необходимости открывать много, да еще вложенных окон, ориентация разработчика снова под угрозой. Поэтому *число дополнительных классов должно быть ограничено* (знаменитый закон психологии «7±2» подсказывает не вводить более 9 классов, что и предусмотрено решающим правилом 1).

Вернёмся к вопросу о связи между метриками трудности D и оценки трудности $\hat{D}$. Коэффициент выборочной корреляции Пирсона на примере выборки 46 модулей Matreshka (результат предоставленный А. И. Паньковской) оказался для них на уровне 0.82. Заметим, что практику интересуют не только, а иногда не столько абсолютные значения трудности модулей. Так D классификация основана на порядковых статистиках явно, а с увеличением числа $\hat{D}$ классов и для этой метрики роль абсолютной величины межклассовых порогов стирается. В виду этого мы предпочли рассматривать ранговую корреляцию.

На модулях рассмотренной выше выборки 74 модулей системы Matreshka коэффициент ранговой корреляции Спирмена ρ между оценкой трудности $\hat{D}$ и трудностью D составляет $\rho = 0.336$ (S= 44842). Поэтому наличие положительной ранговой корреляции между такими оценками может быть принято в данном проекте с вероятностью ошибки первого рода (в соответствии с [16, С. 98]), равной

$$\alpha \approx 2(1 - \Phi(0.336 \cdot 8.544)) \approx 0.006 \qquad (26)$$

(примерно на 1%-м уровне значимости). При всей кажущейся определённости вывода, коэффициенты корреляции дают некую осреднённую оценку связи, ничего не гарантируя для отдельного модуля. Важно посмотреть насколько соответствуют друг другу (или, напротив, смешиваются) классы трудности разных метрик. Весь 5* класс (наиболее трудных в смысле D модулей данной выборки), 71% модулей 4* класса и 57% модулей 3* класса подтверждают свой статус тем, что попадают в 5-й класс - модули с наибольшей оценкой трудности $\hat{D}$. Обратное неверно, но тенденция есть: 80% модулей 5-го класса распределяются в трудных 3*-5* классах, причём 65% - в 4*-5* классах. С другой стороны, из 28 наиболее лёгких в смысле D модулей 71% являются также наиболее лёгкими в смысле $\hat{D}$, но 11% - наиболее трудными (попадают в 5-й класс). В свою очередь, только 7% наиболее лёгких модулей в смысле $\hat{D}$ лежат вне лёгких 0*-2* классов по метрике D, причём только 1 самый трудный из них смог «забраться» в 4* класс. Таким образом, *корреляция между признаками «трудности» и «оценки трудности» реализуется в форме удовлетворительного согласия между соответствующими классификациями. Это не удивительно потому, что Matreshka содержит заметное число явно трудных модулей, на выявление которых в первую очередь нацелены обе метрики*.

Для библиотеки OEM, которая используется в нашей работе в качестве источника контрольных примеров, коэффициент ранговой корреляции Спирмена между оценкой трудности $\hat{D}$ и трудностью D оказывается на порядок меньшим, чем для Matreshka, и равным $\rho = 0.033$ (S= 38399). Теперь нулевую

гипотезу о независимости этих признаков не удаётся отвергнуть ни на каком разумном уровне значимости. Это подтверждается и сравнением классов трудности. Все 6 модулей из класса 5* ненормально трудных по *D* классификации попадают в самый лёгкий 0-й класс по $\hat{D}$. Само по себе это противоречит только наличию положительной корреляции. Однако распределение лёгких модулей 0-го класса (табл. 5) исключает отрицательную корреляцию. Перемешивание классов напоминает «игру случая» и вполне *подтверждает формальный вывод о независимости рассматриваемых признаков трудности в «контрольной» заведомо зрелой библиотеке (OEM).*

Табл. 5 Распределение модулей 0-го класса метрики $\hat{D}$ по классам D метрики (OEM)

D класс	0*	1*	2*	3*	4*	5*
Частота	16	5	2	1	5	6
Частость	0.457	0.143	0.057	0.029	0.143	0.171

Итоговый вывод состоит в том, что статистическая *связь между двумя рассматриваемыми метриками* трудности модулей *может как отсутствовать* (вероятнее, в зрелых проектах), *так и присутствовать* (вероятно, при достаточном числе трудных модулей, поскольку на них нацелены обе метрики).

Фильтрация модулей по классам трудности связана с необходимостью определения модулей, требующих повышенного внимания на завершающем этапе разработки, при поиске дефектов или планировании модификации системы. Поместим все наиболее «подозрительные» модули из рассмотренных выборок в табл. 6,7, руководствуясь подходящими метриками.

Табл. 6 Модули выборки из Matreshka, имеющие признаки максимальной трудности

Модули высших рангов трудности:	*D* класс	$\hat{D}$ класс (ранг)	*q* ранг	**причина ненормальной трудности**
1. matreshka-internals-unicode-ucd-colls.ads	5*	5 (1)	2 несб.	128 громоздких константных агрегатов
2. matreshka-internals-text_codecs-iana_registry.ads	5*	5 (2)	3 несб.	858 конст. агрегатов и константный массив – 858 элементов
3. matreshka-internals-unicode-ucd-core.ads	5*	5 (5)	1 несб.	константный массив – 1354 элемента
4. matreshka-internals-unicode-ucd-core_0000.ads	5*	5 (12)	4 несб.	константный агрегат, имеющий 110 внутренних
5. matreshka-internals-unicode-ucd-cases.ads	5*	5 (3)	9 несб.	22 громоздких константных агрегата
6. matreshka-internals-unicode-ucd-core_000a.ads	5*	5(15)	5 несб.	константный агрегат, имеющий 88 внутренних
7. matreshka-internals-unicode-ucd-norms.ads	5*	5 (2)	8 несб.	50 разных константных агрегатов
A. matreshka-internals-unicode-ucd-indexes.ads	4*	5 (7)	15 несб.	константный массив – 1326 элементов

Табл. 6 составлена на основе класса ненормально трудных модулей по метрике D. Все эти 7 модулей оказались также ненормально трудными и по метрике $\hat{D}$. Поскольку ненормально трудных по $\hat{D}$ значительно больше (21), в таблице (в скобках) уточняются их рейтинги: 1 – самый трудный в смысле $\hat{D}$, 2 – следующий по трудности и т.д. Все эти модули также оказались энергетически несбалансированными (или принадлежат несбалансированной паре, состоящей из описания и тела одного и того же библиотечного модуля). Показан также (под индексом «А») самый трудный модуль из класса 4*: у него и только у него параметры мало отличаются от модулей из «топ 7» труднейших по метрике D.

Все модули таблицы при содержательном рассмотрении обнаруживают специфичесные громоздкие структуры, которые, несмотря на признаки квазирегулярности, при своих размерах «в ручную» *не контролируемы*. Например, модуль под индексом 1 имеет размер свыше 3 Мб (почти 76 тыс. строк). Если такой модуль по какой-то причине будет подлежать модификации или возникнет подозрение на дефект, это потребует особого внимания.

В случае OEM при отборе модулей, для которых прогнозируются риски, связанные с трудностью их понимания, мы не можем взять за основу один, наиболее строгий признак в силу установленной выше некоррелированности

Табл. 7 Модули выборки из OEM с максимальными рисками по причине трудности

Модули с наибольшим классовым весом:	D класс	$\hat{D}$ класс	q класс	**причина относитель-но высок.трудности**
1. oem-com-types.ads	3*	5	1	377 типов/подтипов, 145 констант,25групп
2. oem-com-variant.adb	3*	5	1	939 строк, 2-й по размерам, 21 операция
3. oem-com-com_interface.adb	3*	5	1	716 строк, из крупнейших, 38 операций
4. oem-com-errors.adb	4*	2	2	268 строк, 14 операций
5. oem-com-errors.ads	5*	0	2	находится в контексте oem-com-types. ads
6. oem-com-events-event_object.ads	5*	0	2	находится в контексте oem-com-types. ads и др.
7. oem-com-initialize.adb	5*	0	2	находится в контексте oem-com-types. ads
8. oem-sockets-naming.adb	2*	4	1	545 строк, 39 операций
9. oem-sockets-thin.ads	0*	5	2	469 строк, 23 типа, 49 операций
10. oem-sockets-utils.adb	5*	0	2	находится в контексте oem-sockets-thin. ads
11. oem-com-bstr.ads	5*	0	2	находится в контексте oem-com-types. ads
12. oem-com-create-gc_interface.adb	3*	2	2	234 строки, 13 операций
13. oem-gwindows-events.ads	5*	0	2	находится в контексте oem.gwindows.base. ads

рассматриваемых метрик. При отборе для табл. 7 использовался классовый вес

$$C = C_D + C_{\hat{D}} + C_q \ , \qquad (27)$$

где C_D - номер D класса трудности (напр., $C_D = 5$ для класса 5*);

$C_{\hat{D}}$ - номер $\hat{D}$ класса трудности (напр., $C_{\hat{D}} = 5$ для класса 5-го класса);

$C_q =$ 2 для класса несбалансированных, $C_q = 1$ для только лишь сбалансированных и $C_q = 0$ для хорошо сбалансированных модулей.

Оказалось, что максимальный вес, равный 9 имеют только 3 модуля, один – вес 8 и девять – вес 7. Содержательный анализ исходных текстов показал, что риски, связанные с трудностью понимания, можно действительно усмотреть для модулей в позициях 1-3 табл. 7, в меньшей мере для модулей в позициях 9,8 и в ещё меньшей – 4-12. На остальные 6 отфильтрованных модулей метрики реагировали тоже «законно»: эти тексты разрабатывались с использованием описаний, заимствованных из крупных пакетов, и понимание данных модулей испытывает риск, поскольку эти заимствования потенциально могут быть весьма многочисленны. Только рассмотрение по смыслу позволило в данном случае убедиться, что заимствования малочисленны, и связанный с этим риск невелик. *В табл. 7 попали все модули выборки, которые можно заподозрить в повышенной трудности*. Следовательно, и в данном случае *фильтрация по нашему методу себя оправдала*. Отметим, что при этом в разряд подозрительных, в конечном итоге, попали все ненормально трудные модули в смысле метрики энергетического анализа D, тогда как в смысле $\hat{D}$ – две трети.

8 Заключение

Обоснована целесообразность сохранения в решающих правилах известных порогов для классической метрики оценки трудности Холстеда при условии введения дополнительных классов трудности.

Для метрики трудности энергетического анализа, более современной и реализующей понятие трудности альтернативным способом, разработана новая (относительная) классификация модулей программы. Она позволяет сравнивать модули одной программной системы и, при определённых условиях, трудность разных систем.

Уточнена природа связи между признаками трудности на основе рассмотренных метрик. В новых проектах, где велик процент трудных модулей, на большинство из них указывают значения обеих метрик, обеспечивая удовлетворение критериев наличия статистической связи. Тогда для того, чтобы «вылавливать» безусловно трудные модули, можно использовать любую из них. Чем более зрелой является изучаемая система, чем меньше там трудных модулей и ниже степень их трудности. В этом случае проявляется независимость метрик (вытекающая из различия их природы). Тогда для выделения трудных модулей целесообразно использовать обе метрики и привлекать для контроля другие (в данной работе в этой роли была полезеной метрика нормированного информационного тепла).

В анализе двух конкретных систем обнаружилось, что большую избирательность в отношении трудных модулей имеет метрика трудности энергетического анализа. Важно проверить это предположение на материале других программных систем.

В дальнейших исследованиях разработанные методы и правила целесообразно испытать в масштабах полной программной системы.

Подводя итоги, подчеркнём, что при всей важности метрических оценок тех или иных сторон качества программ, особенно в автоматизированных системах поддержки управления качеством, нельзя забывать о границах их применимости. Мы можем с большой вероятностью предсказать, какие модули программы потребуют больших затрат при своей модификации в виду большей трудности. Однако какие именно модули с высокой вероятностью придётся модифицировать при развитии системы – изученные метрики выявить не помогут. Аналогично сравнительная прогностическая оценка зрелости систем сама по себе ещё не позволяет сравнивать их пользовательское качество. В связи с этим отметим, что представленное здесь исследование является частью процесса комплексного изучения и внедрения систем фреймворков Ада программирования на кафедре моделирования систем и технологий Харьковского национального университета имени В. Н. Каразина. Планируется сопоставлять рассмотренные здесь и другие выводы, основанные на метрических характеристиках с применениями исследуемых систем в учебном процессе.

В заключние автор выражает благодарность коллегам по сообществу Ада программистов В. М. Годунко, С. И. Киркорову, А. И. Паньковской, от которых он получал содействие своей работе.

ЛИТЕРАТУРА

1. Мищенко В. О. CASE–оценка критических программных систем. Том 1. Оценка качества. / В. О.Мищенко, О. В.Поморова, Т. А. Говорущенко ; под ред. Харченко В. С. – Х : Нац. аэрокосмический ун–т «Харьк. авиац. ин–т», 2012. – 201 с.
2. Мищенко В. О. Энергетический анализ программного обеспечения с примерами реализации для Ада-программ / Виктор Олегович Мищенко. – Х.: ХНУ имени В. Н. Каразина, 2007. – 129 с.
3. 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.
4. Shaw Wade H. A Software Science Model of Compile Time / Wade H. Shaw Jr., James W. Howatt, Robert S. Maness, Dennis M. Miller // IEEE Transactions on Software Engineering. – 1989. – Vol. 15, № 5. – P. 543–549.
5. Mishchenko V. O. Does The Different Definitions Of Ada Program Tokens Have Significant Difference? / V. O. Mishchenko // Радиоэлектронные и компьютерные системы. – 2008. – № 7 (34) – С. 103–106.

6. Холстед М. Х. Начала науки о программах / М. Х. Холстед; пер. с англ. В. М. Юфа. – М.: Финансы и статистика, 1981. – 128 с.
7. Shen V. Y. Software Science Revisited: A Critical Analysis of the Theory and Its Empirical Support / V. Y. Shen, S. D. Conte, H. E. Dunsmore // IEEE Transactions on Software Engineering. – 1983. – Vol. SE-9, № 2. - P. 155-165.
8. Генералов К. А. Анализ эффективности использования генетических алгоритмов в задачах при использовании различных языков программирования // Вопросы современной науки и практики. Университет им. В.И.Вернадского. №2(12). 2008. Том 2. С. 148-154.
9. 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.
10. 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.
11. Литвинов Д. Н. Применение энергетических метрик для оценки использования ASIS и в других подобных задачах / Д. Н. Литвинов, В. О. Мищенко // Радіоелектронні і комп'ютерні системи. – 2012. – № 7. – С. 301-306.
12. 982.1-2005 - IEEE Standard Dictionary of Measures of the Software Aspects of Dependability. - Institute of Electrical and Electronics Engineers, 2006. – 34 p.
13. Matreshka [Электронный ресурс] / Vadim Godunko, IE //01.04.2014. – Режим доступа: http://forge.ada-ru.org/matreshka
14. Ada Software – Free DownLoad Ada OEM-2009/2010/2011/2012 library [Электронный ресурс] / MediaScan.by // 15.10.2014. – Режим доступа http://www.mediascan.by/index.files/Page695.html .
15. Hamer Peter G. M.H. Halstead's Software Science – a critical examination / Peter G. Hamer, Gillian D. Frewin // International Conference on Software Engineering. Proceedings of the 6th international conference on Software engineering. - Tokyo, 1982. – P. 197-206.
16. Большев Л.Н. Таблицы математической статистики / Л. Н. Большев, Н. В. Смирнов. – М.: Наука. Главная редакция физико-математической литературы, 1983. – 416 с.
17. GNAT Programming Studio - Википедия Matreshka [Электронный ресурс] http://ru.wikipedia.org/wiki/GNAT_Programming_Studio // 15.10.2014. – Режим доступа http://www.mediascan.by/index.files/Page695.html

Надійшла у першій редакції 23.10.2014, в останній - 03.11.2014.