УДК 519.681.2+004.412.2

Универсальная стратегия подсчета холстедовских примитивов программных модулей: проверка точности и автоматизация алгоритма

Т. А. Ащаулова, В. О. Мищенко
Харьковский национальный университет имени В.Н. Каразина, Украина

Целью работы является применение энергетического анализа в управлении качеством программных систем, разработка универсальных стратегий подсчета основных метрик качества программного обеспечения, разработка программного обеспечения для автоматизации полученных алгоритмов и стратегий. В статье продемонстрированы результаты численного компьютерного эксперимента, которые доказывают эффективность применения разработанного программного обеспечения для подсчета холстедовских примитивов программных модулей.

Ключевые слова: *энергетический анализ, примитивы Холстеда, программное обеспечение, компьютерный эксперимент, спецификационная энергия, универсальная стратегия подсчета.*

Метою роботи є застосування енергетичного аналізу в управлінні якістю програмних систем, розробка універсальних стратегій підрахунку основних метрик якості програмного забезпечення, розробка програмного забезпечення для автоматизації отриманих алгоритмів і стратегій. В статті продемонстровані результати чисельного комп'ютерного експерименту, котрі доводять ефективність застосування розробленого програмного забезпечення для підрахунку холстедівських примітивів програмних модулів.

Ключові слова: *енергетичний аналіз, примітиви Холстеда, програмне забезпечення, комп'ютерний експеримент, специфікаційна енергія, універсальна стратегія підрахунку.*

It is often difficult to give a formal definition of programming symbols (semantic atomic units of software); therefore the special applications, which implements automatic counting, is used to solve this problem. The article presents the results of a numerical computer experiment which proves the efficiency of the software developed for counting the Halstead's primitives of program modules.

Key words: *energy analysis, Halstead's primitives, software, computer experiment, specification energy, universal counting strategy.*

1. Общая постановка задачи и основные понятия

Целью данной работы является применение энергетического анализа в управлении качеством программных систем, разработка универсальных стратегий подсчета основных метрик качества программного обеспечения, разработка программного обеспечения для автоматизации полученных алгоритмов и стратегий. *Стратегией подсчета* (counting strategy) для исходных текстов программ является метод определения примитивных характеристик (примитивов) модулей программ, в частности *длин* и *словарей*. М. Холстед [1] разработал ряд метрик, в связи с которыми были разработаны стратегии подсчета. Дальнейшим развитием стратегий подсчета можно считать подходы *энергетического анализа* программных систем [2, 3]. Наука о программах Холстеда - одна из первых теорий качества программного обеспечения, в которой были введены метрики *объема* (модуля программы), *трудности*

(модуля) и *усилий* (необходимых для разработки модуля). Используя данные метрики, можно с какой-то вероятностью прогнозировать число допущенных ошибок программирования, трудность их отыскания и трудность модификации модуля, затраты времени на разработку отдельного модуля или их системы [4].

Энергетический анализ обобщает понятие объема и трудности взаимосвязанных модулей многомодульных программ (метрики *объёма разработки* и *трудности разработки*), вводит *спецификационную энергию* (метрика интерфейсных модулей и программной системы в целом), определяет *работу программирования* (метрика, обобщающая метрику усилий Холстеда), *интеллектуальное тепло* и уравнение типа *первого начала термодинамики*.

Длина модуля программы – это количество программных символов (tokens в терминологии Холстеда), на которые разбивается исходный текст данного модуля, а словарь – число разных программных символов, которые встречаются в этом тексте. При этом далеко не всегда программными символами могут считаться отдельные лексемы данного языка программирования (lexical tokens), часто это комбинации слов и знаков (нередко разделённые другими лексическими элементами, которые в их данном употреблении лишены смысла в отрыве друг от друга (знак «больше или равно», пары скобок разного вида, в некоторых языках такие конструкции как *if … else* или *while (…))*

Исчерпывающе точное определение программных символов (смысловых атомарных единиц компьютерных программ) данного языка программирования требует существенных затрат квалифицированного труда и, отчасти, может оказаться основанным на субъективных решениях.

Существуют, однако, методы, которые можно было бы назвать недетерминированными стратегиями, которые позволяют производить приближенные подсчёты длин и словарей, не используя точных определений программных символов [2, 5].

В данной статье разрабатывается метод очистки программных файлов для дальнейшего подсчета спецификационной энергии программной системы, а также рассматривается универсальный метод подсчета холстедовских метрик с привлечением внешнего эксперта, изложены результаты сравнительного анализа полученных оценок основных метрик качества с точными значениями данных метрик (были получены путем прямого подсчета программных символов), а также разработано программное обеспечение для автоматизации изложенной в статье стратегии с привлечением внешнего *эксперта*.

Рассматриваемый метод не зависит от языка программирования, однако, для его применения эксперт должен обладать знаниями, позволяющими ему определять лексические единицы языка.

Метод может быть использован не только в качестве получения оценочных значений величин метрик, но и как самостоятельная схема подсчета холстедовских метрик.

2. Использование программных символов при реализации стратегии подсчета

Программные символы (tokens) являются более крупными образованиями в исходных текстах программ, чем лексические атомарные единицы (lexical

tokens), но более мелкими, чем операторы (statements), команды или инструкции ЯП (instructions). Например, в следующем фрагменте исходного текста на языке C/Java/C#/C++: «int [] A = {1, 2, 3, 4}; int b = 4;».

программными символами являются:

1) int 2) [ ] (вместе!) 3) A 4) = 5) { … } (вместе!) 6-9) 1 2 3 4 10) , 11) b.

Итого, 11 разных программных символов, а всего использовано 13. Значит, по определению: словарь n = 11, длина N = 13.

Одна из наиболее востребованных холстедовских метрик – «оценка трудности» требует еще разделения использованных программных символов на операторы (operators) и операнды (operands). Операторы играют, полностью или частично, служебную роль (например, скобки), а операнды наполняют программу конкретным содержанием, их конкретный вид не влияет на понимание смысла программы (но влияет на результат её работы, выход). В данном фрагменте операндами являются только A, b, 1, 2, 3, 4. Поэтому словарь и длина разбиваются так: словарь операторов n1 = 6, словарь операндов n2 = 11-6 = 5, операторная часть длины N1 = 11, часть операндов в длине N2 = 13.

В данной статье рассматривается универсальный метод подсчета программных символов модуля – метод с привлечением внешнего эксперта. Значения, полученные данным методом, будут сравниваться с точными значениями подсчитываемых параметров.

Учитывая приближенный характер оценок метода с привлечением эксперта, имеет смысл применить *программную диверсность* [6], то есть использовать для анализа выборки сразу два метода – точный и с привлечением эксперта, а в качестве признака точности – разность между их результатами. Если величина разности не превосходит некоторой приемлемой величины, например, 5-10%, то полученным результатам можно верить. В качестве оценки можно взять любой из примененных методов или же среднее значение. В случае наблюдения существенного расхождения – сообщаем об этом факте пользователю, чтобы он принял решение какому методу доверять. В случае значительных расхождений можно применить ручную проверку или же использовать третий метод.

3. Подсчет спецификационной энергии программ

Спецификационная энергия программной системы может быть определена по следующей формуле: $E = E(V^*, \lambda) = {V^*}^3 / \lambda^2$, где V^* – *потенциальный объем* программной системы, агрегирующий потенциальные объемы модулей конкретной программной системы; λ – *уровень языка программирования*, на котором написаны программные модули рассматриваемой программной системы.

Такой параметр качества программной системы, как спецификационная энергия, отражает зрелость как самой программной системы, так и опытность разработчиков, принявших участие в написании кода.

Однако, прежде чем приступить к подсчету спецификационной энергии программных модулей, нужно предварительно обработать файлы, таким образом подготовив почву для дальнейшего анализа.

Для языка Ada был разработан и опробован алгоритм очистки (предварительной подготовки) файлов к дальнейшей обработке с целью подсчета интересующих метрик. Основные пункты данного алгоритма изложим ниже.

Анализ программной системы, написанной на языке программирования Ada, начинаем с .ads-файлов – файлов спецификации, содержащих объявления переменных, типов данных и подпрограмм, затем анализируем .adb-файлы, которые содержат уже непосредственно реализацию описанного в спецификационных файлах функционала.

Далее сканируем список подключенных в файле библиотек. Подключенные Ada-библиотеки удаляем. Для пользовательских библиотек, подключенных в файле, оставляет только инструкции с ключевым словом with, use-инструкции удаляем. Очищаем файл от всех комментариев – однострочных и многострочных.

Удалению подлежат также строки, содержащие объявления типов и подтипов: блоки, начинающиеся с ключевых слов type, subtype. За исключением видимой части пакета, в других местах удаляем объявленные объекты – константы, переменные.

При разборе .adb-файла для локальных функций (описания которых нет в подключенных библиотеках и в соответствующем файле .ads) оставляем только название функции и тип возвращаемого результата в виде:

local function xx(t:Long_Float) return vector;

Сопровождаем указанную строку комментарием --*local* для упрощения дальнейшего анализа очищенных файлов.

Если внутри локальной функции присутствуют операции ввода/вывода, то такие операции оставляем в теле функции, остальные участки кода удаляем.

Функции, которые описаны в соответствующем анализируемому .adb-файлу .ads-файле, удаляем из .adb-файла, если внутри них нет операций ввода/вывода.

Если внутри таких функций есть операции ввода/вывода, тогда в файле .ads данные функции сопровождаем комментарием --*parameters in body* для упрощения дальнейшего анализа очищенных файлов; внутри .adb-файла такие функции сопровождаем комментарием --*non-local*. При этом удаляем все операции из тела функции, кроме операций ввода/вывода.

После того, как все файлы очищены по указанному алгоритму, производится подсчет спецификационной энергии программной системы. Указанный алгоритм может быть применен только к программным системам, написанным на языке программирования Ada, что связано с привязкой такого рода алгоритмов непосредственно к синтаксису языка. Однако, в дальнейшем данный алгоритм может быть расширен на другие языки программирования путем изучения их спецификаций, а также корректного определения таких понятий, как *модуль*, *группа* и *блок*.

Например, для языка программирования C# данные понятия могут быть определены следующим образом:

Модуль – всякий компилируемый модуль, который является описанием функции в интерфейсе (или описанием с модификатором abstract в абстрактном классе) или же является непосредственной реализацией - телом - функции.

Группа – класс, объемлющий функциональные модули. При условии, что один класс представляет собой один файл программы – исходя из требований *Coding Conventions* и хорошего тона программирования.

Блок – собственный блок модуля – исходный текст данного модуля; явными блоками являются описания и тела функций без входов; описания и тела входов функций.

4. Алгоритм метода оценки с привлечением внешнего эксперта

Данный алгоритм позволяет использовать универсальную стратегию подсчета программных символов и основных метрик качества программного обеспечения и является основным алгоритмом, положенным в основу разработанного программного продукта, речь о котором будет идти в следующем разделе данной статьи.

После того, как определены файлы, которые будут обрабатываться, программа читает первый по списку файл и проходит его от начала до конца с целью отсеять из рассмотрения строки чистых комментариев и пустые строки. Вид комментариев зависит от выбранного языка программирования, на котором созданы анализируемый программы. Для возможности дальнейшего анализа файла составляется список *содержательных строк*, которые содержат код и, возможно, также комментарии. Слитные массивы строк чистых комментариев подсчитываются (это, по определению, программные символы типа «комментарий»). Альтернативой является переписывание файла с удалением из него всех не содержательных строк. В таком случае имеет смысл удалить вообще все комментарии, строковые и символьные литералы (разумеется, подсчитывая их число для последующего учёта в результатах анализа).

В результате первого прохода исходного кода получаем число L всех содержательных строк и список диапазонов (т.е. массивов) содержательных строк в файле (которые разделяются пустыми строками и/или строками чистых комментариев).

Планируется выборка содержательных строк. Её объем N при малых L совпадает с этим числом. При «наиболее распространённых» значениях L, т.е. примерно от 100 до 500, объём выборки должен быть около 10, а с ростом L он, естественно растёт, но «медленно» (логарифмически). Номера выбираемых строк в пределах от 1 до L определяются с помощью датчика случайных чисел. После этого файл во второй раз проходится от начала к концу.

В окно программы для анализа человеком-оператором последовательно выводятся все содержательные строки файла из намеченной выборки, которая, как уже было описано ранее, производилась из содержательных строк файла случайным образом, что позволяет в дальнейшем анализировать параметры данной выборки и говорить уже о закономерностях в генеральной совокупности. Оператор вводит указатели на концы всех программных символов (или их связные части) которые он распознаёт в каждой строке в версии программы,

написанной на Ada, или же вводит количество программных символов, которые встретились в выданной строке – а также количество операндов.

Например, пусть выводится строка: *abraCodabrasic (*

(Вместе с ориентирами для облегчения определения номеров символов человеком):

abraCodabrasic (
1234567890123456 78

Оператор, вероятно, ответит на это вводом строки:

4 -12 16 18

где знак минус означает, что конец принадлежит символу-операнду, а не оператору (так, например, будет, если *abra* и *sic* – ключевые слова данного формального языка, а *Codabra* – идентификатор, выбранный программистом);

знак *(*, возможно, является частью программного символа-оператора «пара скобок»;

пробельные символы при последующей обработке программных символов могут быть автоматически отброшены, так что в нашем примере вместо позиции 4 можно с тем же успехом указать 5.

Каждое такое сообщение оператора позволяет нарастить счётчик программных символов и счётчик операторов (явно поддерживать счётчик для операндов не обязательно). При этом, если оператор считает, что на строке представлена только часть программного символа, например, только одна из пары скобок, он может это уточнить. В нашем примере наряду с указанием просто номера 18 концевого символа присутствующей на строке части программного символа «пара скобок» можно ввести комбинацию:

*18*1/2*

где в качестве числителя и знаменателя могут фигурировать только одиночные десятичные числа, определяемые одной цифрой. (Данная возможность в первых версиях автоматического анализатора может опускаться).

По окончании второго прохода число программных символов («длина») в данном исходном тексте некого модуля СПС – *схемы программной системы*, оценивается по формуле в зависимости от того, принимаются ли во внимание комментарии: $Length_0 = (P/N)L$, $Length = (P/N)L + C$, где P – суммарное число программных символов данной выборки строк файла; N – объём выборки (число проанализированных строк); L – число содержательных строк; C – число слитных массивов строк комментариев.

Комментарии следует учитывать, причём изолированные массивы комментариев следует считать разными воплощениями одного и того же программного символа.

Аналогично оценивается общее число операторов в данном тексте (комментарии к числу операторов не относятся, но считаться операндами тоже могут лишь условно: желательно, чтобы при автоматическом анализе свойство «быть комментарием» проверялось, и счётчик комментариев наращивался вместо «счётчика» операндов).

Далее выполняется оценка словаря – числа различных использованных программных символов. Это можно сделать, но, конечно, только весьма

приближенно, исходя из гипотезы выполнения «уравнения длины программы» Холстеда: $Length\hat{} = n_1 log_2 n_1 + n_2 log_2 n_2$, где n_i – словарь операторов (i = 1) или операндов (i = 2).

Значение $Length\hat{}$ случайным образом колеблется вокруг $Length_0$.

Для получения числовой оценки словари операторов и операндов следует связать, например, по гипотезе выполнения уравнения длины программы. Есть смысл также огрубить соотношение и свести нахождение решения к решению уравнения следующего вида, которое можно использовать в подсчетах, проводимых непосредственно в программном продукте, реализующем стратегию подсчета с привлечением внешнего эксперта: $Length_0 = Vocabluary * log_2(Vocabluary)$.

5. Реализация метода оценки подсчета холстедовских примитивов с привлечением внешнего эксперта

Для того, чтобы обработка файлов, содержащих исходные анализируемые коды программ, проходила в удобном для эксперта режиме, было разработано программное обеспечение, автоматизирующее подсчет таких параметров программы, как длина программы, словарь, а также количество операторов и операндов.

При запуске реализующей данный метод программы, пользователю нужно определиться с языком программирования, который использовался для создания выбранного для анализа файла. Для разбора таких файлов привлекается эксперт, обладающий соответствующим уровнем знаний в выбранном языке, так как полученный результат для всего файла будет зависеть от компетентности эксперта и его умении правильно определять лексемы программы – операторы и операнды.

Каждый язык программирования содержит в себе отличительный набор операторов и операндов, которые, по сути, и составляют уникальный словарь языка, а, следовательно, и написанной на нем программы. Поэтому так важно привлекать к подсчету и анализу основных метрик компетентных экспертов.

Перед запуском программы один или же несколько файлов, которые будут проанализированы, должны находиться в одной директории с исполняемым файлом программы, что достигается путем перемещения файлов в корневой каталог проекта – ExpertCounting.

Программа была реализована на двух языках программирования – Ada и Java. Для того, чтобы программа подсчета холстедовских примитивов с применением стратегии с привлечением внешнего эксперта могла быть включена в уже существующий и ранее созданный программный продукт для подсчета программных символов и метрик по другим алгоритмам, данная программа была написана на языке программирования Ada. Однако, основным недостатком языка Ada является ограниченная переносимость написанных на нем программ, следовательно, было принято решение (и успешно реализовано) создать программный продукт на языке Java – кроссплатформенном языке программирования, кроссплатформенность которого достигается за счет наличия *виртуальной машины Java* – программы, которая является прослойкой между операционной системой и Java-программой [7].

После запуска программы пользователем, вызывается операция readFile модуля Counting, который просматривает файл и изменяет переменные количества значимых строки и строк комментариев на соответствующие значения для указанного файла.

Далее производится генерация случайных номеров строк из числа значимых строк файла, для чего используются операции getFileStrings, getRandomNumbers для генерации и сортировки массива случайных номеров значимых строк файла. Размер этого массива определяется в зависимости от числа значимых строк данного файла.

Оператор получает на экран каждую из строк (по номерам, которые содержаться в описанном выше массиве). Он должен указать количество распознаваемых им программных символов и операндов. Операция getResultsForRandomString используется для подсчета программных символов и записи соответствующей информации о каждой выданной оператору на анализ строке в объект-коллекцию.

После того, как оператором были обработаны все значимые строки, вызывается операция countMetrics, а затем getProgramVocabluary для получения информации об общем числе строк файла, числе строк комментариев, числе значимых строк файла, а также числе операторов и операндов на строку кода, подсчет количества операторов и операндов для всех значимых строк файла, получения мощности алфавита анализируемого файла.

Результатом работы программы является выдача информации о длине программы, словаре программы, а также о количестве операторов и операндов на экран эксперту.

Программа выдает на обработку эксперту весь список файлов, содержащихся в указанном ранее каталоге, в порядке их очереди размещения в каталоге.

6. Вычислительный эксперимент: сравнение значений, полученных двумя способами

Для проведения анализа корректности работы разработанного программного продукта, а также для оценки эффективности стратегии подсчета холстедовских метрик с привлечением внешнего эксперта, было принято решение по обработке 50 программных модулей, написанных на языке MathCAD.

В работе Боровинского А. [8] был введен формальный язык таких программ на базе XML, определенный проекцией языка XMCD-файлов. Воспользовавшись инструментом для обработки таких файлов, написанным Боровинским А., были получены точные значения длины и словаря программы для каждого файла выборки.

В качестве выборки были взяты файлы из [9], данные MatchCAD-файлы написаны экспертом в области программирования для MatchCAD, следовательно, могут быть взяты в качестве программных модулей с оптимальным подходом к их написанию, а, значит, могут быть отнесены к зрелым программным модулям – код которых может служить примером оптимального решения поставленных задач.

В Табл. 1 - сравнительной таблице результатов сравниваемых точных значений и их оценок приведены полученные результаты. N – точное значение величины длины программного модуля; n – точное значение величины словаря программного модуля. Следовательно, M – значение величины длины программного модуля, полученные путем взаимодействия с внешним экспертом посредством разработанного программного обеспечения; m – значение величины словаря программного модуля, полученного тем же способом.

Табл.1. Сравнение результатов метода с применением стратегии подсчета с привлечением внешнего эксперта и точных значений подсчитываемых величин

№ файла	*N*	*M*	*N/M*	*n*	*m*	*n/m*
1	863	998.20	0.86	96	124.78	0.77
…						
23	738	614.09	1.20	68	76.76	0.89
…						
50	3207	2575.00	1.25	146	257.50	0.57
Среднее			1.15			0.82
Коэффициент корреляции	N на M		0.94	n на m		0.65

После обработки полученных результатов для выборки файлов было получено значение среднего отклонение стратегии с привлечением внешнего эксперта от точного результата, которое составляет 6%. При этом стоит отметить, что время, затраченное экспертом на анализ более 50 файлов с применением разработанного программного обеспечения приемлемо и составляет, в среднем, не больше 1 минуты на файл; разработанное программное обеспечение позволяет анализировать таким способом программные модули, написанные на любом языке программирования, который знаком эксперту.

7. Выводы

Разработано программное обеспечение для реализации стратегии подсчета программных символов и основных холстедовских метрик с привлечением внешнего эксперта. Проведен сравнительный анализ полученных результатов с точными значениями для выборки файлов, которые могут быть отнесены к оптимальным примерам решения поставленных задач. Полученное отклонение от точных значений для данной выборки составляет не больше 6%, а страховкой от случайных ошибок является программная диверсность. Разработанная стратегия является универсальной относительно языка программирования, однако, для ее реализации привлекаемый внешний эксперт должен обладать знаниями о синтаксисе языка.

ЛИТЕРАТУРА

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

2. Мищенко В. О. Энергетический анализ программного обеспечения с примерами реализации для Ада-программ [Текст] / В. О. Мищенко. – Х.: ХНУ имени В.Н. Каразина, 2007. – 129 с.
3. Мищенко В. О. Термодинамический подход к моделированию процесса программирования[Текст] // Модерирование и обеспечение систем и технологий, Ч. 1 – Математическое моделирование физических процессов и технических систем. – Х.: ХНУ имени В.Н. Каразина, 2014. – С. 209-260.
4. *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.
5. Диденко Е. В. Обеспечение работоспособности подсчета программных символов на примере языка Ада [Текст] / Е. В. Диденко, В. О. Мищенко.
6. Мищенко В. О. Преимущества, затраты и риски модификации реализации методов дискретных особенностей с целью оптимизации [Текст] / В.О. Мищенко // Вісник Харківського національного університету імені В. Н. Каразіна серія «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління». – Вып. 28, 2015. – С. 69-76.
7. Шилдт Герберт Java. Полное руководство, 8-е издание [Текст] / Герберт Шилдт, пер. с англ. М.: ООО «И. Д. Вильямс», 2012. – 1104 с.
8. Borovinskiy A. A. Comparison of program developed on the universal programming language and using mathematical package (The 4rd Workshop on Ada Technology and Language Diversity Kharkiv, May, 18) / Alexey Borovinskiy, Andrii Gakhov, Viktor Mishchenko // in the “DEpendable Systems, SERvices and Technologies, Ukraine, May 18-23, 2016”.
9. Макаров Е. Г. MathCAD. Учебный курс [Текст] / Е. Г. Макаров. – СПб.: Питер, 2009. – 384 с.

Надійшла у першій редакції 01.05.2017, в останній - 25.05.2017.