Оглавление

Предисловие

Разработка программного обеспечения представляет собой процесс, в который вовлечено множество разработчиков. Непрерывное повышение сложности функций, реализуемых программами в информационных системах, приводит к увеличению их объема и требует большей трудоемкости создания. Соответственно сложности программ возрастает количество выявляемых и остающихся в них дефектов и ошибок, что отражается на их качестве. Достаточно часто разрабатываются комплексы программ объемом в миллионы строк текста, которые принципиально не могут быть безошибочными. Поэтому проблема оценки качества созданных программных средств всегда была и остается очень важной задачей, решение которой позволяет на ранних стадиях жизненного цикла определить характеристики создаваемой программы.

Ряд существующих методик направлены на оценку параметров будущих программ, что позволяет адекватно оценить трудозатраты и стоимость разработки. Многие ошибки, обусловленные неопределенностью или некорректностью технических заданий и спецификаций требований, могут и должны быть выявлены на ранних стадиях проектирования, что способствует его ускорению и повышению качества. Стратегической задачей в жизненном цикле современных информационных систем стало обеспечение качества программных средств и баз данных. Для решения такой задачи необходимо применение специализированных методик, разработанных в различное время. Разнообразие таких методик позволяет применять их на различных стадиях жизненного цикла программ – от начала разработки до сертификации программного обеспечения.

Данное издание посвящено практической части процесса оценки качества программных средств, что актуально при изучении учебной дисциплины «Стандартизация, сертификация и управление качеством программного обеспечения». Разделы книги организованы по тематическому принципу таким образом, чтобы их было удобнее применять в процессе проведения практических занятий, продолжающих процесс изучения курса. В каждом разделе приведены краткие теоретические сведения, касающиеся той или иной методики, рассмотрены примеры решения типовых задач по каждой из них. Предложен набор задач, предназначенных для самостоятельного решения.

Практическое занятие 1 посвящено изучению оценки характеристик программных средств на основе лексического анализа текста программ. Рассматриваются задачи, основанные на применении методик Холстеда, Джилба и Чепина.

В рамках практического занятия 2 рассматриваются приемы оценки структурной сложности программ на основе применения метрики Маккейба.

Практическое занятие 3 позволяет освоить методы оценки характеристик программных средств на основе процедурно-ориентированных метрик, в частности – рассчитывать количество функциональных указателей, оценивать связность и сцепление модулей в программе.

Оценка характеристик программ на основе применения объектно-ориентированных метрик осуществляется в материале практического занятия 4, где рассматриваются задачи на применение метрик Мартина, Чидамбера и Кемерера, Лоренца и Кидда, Абреу.

Практическое занятие 5 посвящено освоению приемов оценки надежности программ на основе применения моделей Джелинского – Моранды, Миллса, Нельсона, эвристической модели.

Материал практического занятия 6 предназначен для тренировки навыков при оценке качества программ на этапе стандартизации.

Практическое занятие 7 предлагает проведение оценки программ на этапе сертификации, т. е. рассматривает процессы, протекающие уже после внедрения программных средств в работу информационных систем.

Книга предназначена, в первую очередь, для студентов специальностей «Программная инженерия» и «Бизнес-информатика» при изучении учебной дисциплины «Стандартизация, сертификация и управление качеством программного обеспечения», для преподавателей при подготовке занятий, а также для разработчиков программного обеспечения, которые хотели бы более подробно изучить методы, применяемые при оценке качества программных средств. В то же время и сложившиеся специалисты, развивающие навыки или применяющие свои знания для разработки и внедрения программного обеспечения, также найдут полезные сведения как в практическом, так и методологическом плане.

Практическое занятие 1

ОЦЕНКА ХАРАКТЕРИСТИК ПРОГРАММ НА ОСНОВЕ ЛЕКСИЧЕСКОГО АНАЛИЗА

1.1. Метрики Холстеда

1.1.1. Теоретические сведения

Особенности формирования словаря программы

Любая программа определяет последовательность действий над операндами с помощью операторов. Исходный текст программы, записываемой на том или ином языке программирования представляет собой набор текстовых строк, которые записываются по специальным правилам, и в том числе имеет свои элементы. *Операнд* – это некоторый объект или величина, обрабатываемая в программе, а *оператор* представляет собой обозначение конкретного действия, выполняемого по отношению к операнду.

Если считать, что словарь любой программы состоит только из имен операторов и операндов, то тексты программ всегда удовлетворяют следующим условиям:

- маловероятно появление какого-либо имени оператора или операнда много раз подряд – языки программирования, как правило, позволяют создавать такие конструкции, в которых подобные фрагменты программы имеют минимальную длину;
- циклическая организация программ исключает многократное повторение какой-либо группы операторов и операндов – более компактные варианты текстов получаются при разумном использовании развитых возможностей языков программирования, причем многообразие языков предоставляет богатую палитру инструментов;
- блоки программ, требующие периодического повторения при ее исполнении, обычно оформляются как процедуры или функции, поэтому в текстах программ достаточно применения только их имен;

• имя каждого операнда должно появляться в тексте программы хотя бы один раз – многие среды программирования обращают внимание программистов на неиспользуемые имена, которые следует удалять из текста программ, чтобы сократить объем памяти, используемой при объявлении переменных.

Схематично эти условия, относящиеся к тексту программ, отображены на рис. 1.1.

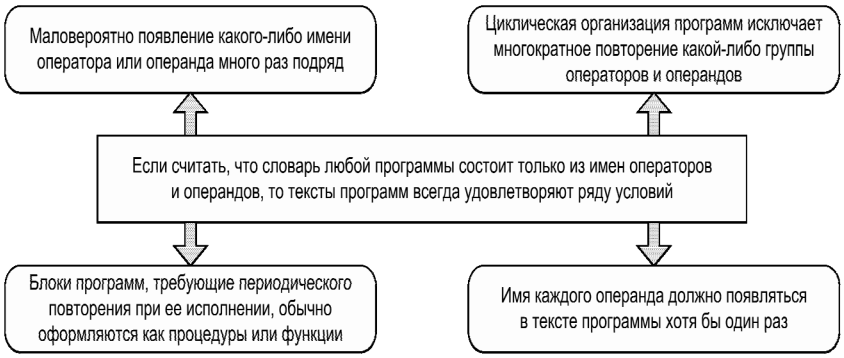

Рис. 1.1. Условия образования словаря программы

В настоящем издании будут в основном рассматриваться примеры программ, разработанных на языках программирования C, C++ и C#, поэтому описание будет проводиться именно для них.

В рассматриваемых языках программирования добавление к набору символов, являющихся выражением, знака «;» превращает его в оператор. Так, набор символов «*i*++» является выражением, а конструкция «*i*++;» уже является оператором (или statement по терминологии Кернигана и Ритчи [3]).

При разработке программ существует ряд особенностей, связанных со скобками: круглыми, фигурными и квадратными.

Одним из действий, которые могут быть записаны в исходном тексте программ, является вызов функций, записываемый в формате

Имя_функции().

В словаре операторов и операций, которые будут формироваться далее в примерах, будем различать вызов функции по появлению пары круглых скобок и следующим перед ними именем. Следует обра-

тить внимание, что при этом нужно отличать пару круглых скобок, используемых с другими языковыми конструкциями (например, с операторами *if* или *for*).

При формировании словаря операторов и операций учитываются еще и фигурные скобки. Они, казалось бы, не вызывают никаких действий, но связаны с ограничением области видимости переменных.

Операции, которые связаны с квадратными скобками, это обращение к элементам массива. Действия, связанные с их интерпретацией, обеспечивают доступ к элементам массивов.

Операторы и операции связаны с объектами, над которыми они выполняются. Таким объектами являются, прежде всего, константы, простые переменные, массивы и структуры, однако подобными объектами (операндами) являются и функции, а также классы и методы.

Сделаем еще одно важное замечание. При формировании словарей будем учитывать элементы, используемые в разделах описаний, поскольку эти элементы также можно рассматривать как операнды или операторы.

Измеряемые свойства программ

При разработке программы формируется ее текст на каком-либо языке программирования, реализующий алгоритм получения искомого результата на основании обработки заданной совокупности данных. В тексте программы можно идентифицировать все операнды, определенные как переменные или константы, используемые в данной реализации. Аналогичным образом идентифицируются все операторы, определенные как символы или комбинации символов, влияющие на способ обработки, изменение значения или порядок следования и преобразования операндов. Исходя из идентификации операторов и операндов, можно определить ряд измеримых категорий, обязательно присутствующих в версиях любого алгоритма. Они определяются метриками, с помощью которых могут быть получены основные характеристики качества программ.

В состав измеримых свойств любого представления алгоритма (или программы) могут быть включены следующие метрические характеристики:

- η_1 – число простых (уникальных) *операторов,* появляющихся в данной реализации;

- η_2 – число простых (уникальных) *операндов,* появляющихся в данной реализации;
- N_1 – общее число всех *операторов,* появляющихся в данной реализации;
- N_2 – общее число всех *операндов,* появляющихся в данной реализации;
- f_{1j} – число появлений в программе j-го оператора, где j = 1, 2, 3, ..., η_1;
- f_{2j} – число появлений в программе j-го операнда, где j = 1, 2, 3, ..., η_2.

Учитывая эти основные метрические характеристики для программы, в конкретной реализации текста программы можно определить:

- словарь $\eta = \eta_1 + \eta_2$;
- длину реализации программы $N = N_1 + N_2$;
- длину программы $\tilde{N} = (\eta_1 \cdot \log_2 \eta_1) + (\eta_2 \cdot \log_2 \eta_2)$.

Следует отметить, что помимо своего прямого назначения метрики длины программы и длины реализации можно использовать для выявления *несовершенств программирования*, которые являются следствием применения не самых удачных приемов программирования. Если расчетные значения длины программы и длины реализации отличаются более чем на 10 %, то это свидетельствует о возможном наличии в программе следующих шести *классов несовершенств*:

1. Наличие последовательности дополняющих друг друга операторов к одному и тому же операнду, например $A + C - A$. Понятно, что в подобном случае будет выполнено два совершенно ненужных действия, дополняющих переменную C одной и той же величиной, взятой с противоположными знаками.

2. Наличие неоднозначных операндов, например $A = D$ и $A = C$. При выполнении таких действий программа будет поставлена в затруднительное положение, поскольку присвоение осуществляется путем приравнивания значения операнда, указанного в левой части, значению, приведенному в правой части выражения. В лучшем случае произойдет ненужное присвоение нового значения уже имеющемуся.

3. Наличие синонимичных операндов, например $A = B$ и $C = B$. Поскольку одно и то же значение должно быть присвоено разным переменным, то для данного примера переменная B вообще может не использоваться. Более лаконичным вариантом является простое приравнивание значений переменных A и C.

4. Наличие общих подвыражений, например:

$$(A + B) \cdot C + D \cdot (A + B).$$

Здесь применено совсем не обязательное повторение суммирования переменных A и B, что приводит к дополнительному времени выполнения программы.

5. Ненужное присваивание, например $C = A + B$, если переменная C используется в программе только один раз. При однократном выполнении каких-либо операций над переменной нецелесообразно вводить дополнительный операнд, это ведет к увеличению объема памяти, резервируемой под переменные программы, и увеличивает размер словаря.

6. Наличие выражений, которые не представлены в свернутом виде как произведение множителей, например:

$$X \cdot X + 2 \cdot X \cdot Y + Y \cdot Y.$$

Данное преобразование можно представить как

$$(X + Y) \cdot (X + Y),$$

т. е. свернуть выражение до квадрата суммы переменных X и Y.

Такое представление окажется более лаконичным и сократит время, необходимое для выполнения программы.

В соответствии с приведенными определениями применяются следующие соотношения:

$$N_1 = \sum_{j=1}^{\eta} f_{1,j}; \tag{1.1.1}$$

$$N_2 = \sum_{j=1}^{\eta} f_{2,j}; \tag{1.1.2}$$

$$N = \sum_{i=1}^{2} \sum_{j=1}^{\eta} f_{i,j}. \tag{1.1.3}$$

Таким образом, длина реализации и объем программы определяются исключительно на основе анализа текста программы путем подсчета количества операндов и операторов, а также числа их вхождений в текст программы, т. е. на основе лексического анализа текста программы. Длина программы представляет собой математическое ожидание количества слов в тексте программы при фиксированном словаре.

Другой важной характеристикой программы является ее объем V. В отличие от длины программы N объем измеряется не количеством слов, а числом двоичных разрядов. Если в словаре имеется η слов, то для задания номера любого из них требуется минимум $\log_2 \eta$ бит.

Объем программы определяется следующим образом:

$$V = N \cdot \log_2 \eta = \eta \cdot \log_2^2 \eta . \quad (1.1.4)$$

Тогда с точностью до обозначений полученные соотношения окажутся совершенно идентичными, хотя смысл их будет различным:

$$N \approx \eta \cdot \log_2 \eta ; \quad (1.1.5)$$

$$V = \eta \cdot \log_2^2 \eta . \quad (1.1.6)$$

В первом случае зафиксирована взаимная связь между длиной программы N и размером словаря η, во втором – между величиной словаря и объемом программы V. Следовательно, по известному размеру словаря η можно найти значения N и V. Идентичность этих выражений говорит о том, что соотношение между величиной словаря и длиной текста единственно и взаимно однозначно.

Выше было отмечено, что словарь программы состоит только из операторов и операндов. Учитывая принятые обозначения, соотношение Холстеда примет следующий вид:

$$N = \eta_1 \cdot \log_2 \eta + \eta_2 \cdot \log_2 \eta \approx \eta_1 \cdot \log_2 \eta_1 + \eta_2 \cdot \log_2 \eta_2 = N_1 + N_2 . \quad (1.1.7)$$

Как правило, при проведении статистических исследований текстов программ к словарю операторов относят следующие элементы:

- имена арифметических и логических операций;
- присваивания;
- условные и безусловные переходы;
- разделители;
- скобки (парные);
- имена процедур и функций;
- выражения типа BEGIN...END, IF...THEN...ELSE, DO...WHILE.

Выражения типа BEGIN...END, IF...THEN...ELSE, DO...WHILE и им подобные, осуществляющие блочную группировку операторов, при этом рассматриваются как единые операторы (то же относится и к парам скобок).

Величины количества операторов и операндов η_1 и η_2 независимы и могут принимать произвольные значения. Однако этого нельзя сказать относительно N_1 и N_2, т. е. числа появления всех операторов $N_1 = \eta_1 \cdot \log_2 \eta_1$ и всех операндов $N_2 = \eta_2 \cdot \log_2 \eta_2$ в тексте программы: между ними можно установить приблизительное соответствие, причем оно будет взаимно однозначным. В каждом конкретном случае каждый операнд не может позиционироваться в программе обособленно, он входит в текст, по крайней мере, хотя бы с одним оператором: например, с разделителем (точка с запятой), отделяющим его от других операторов, или другим набором символов, определяющим способ действия над этим операндом. В то же время применение нескольких операторов к одному операнду маловероятно. Поэтому можно утверждать, что $N_1 \approx N_2$, хотя величины словарей η_1 и η_2 могут сильно отличаться друг от друга. Это позволяет прийти к весьма важному практическому выводу относительно объема программы:

$$N \approx 2 \cdot N_2 = 2 \cdot \eta_2 \cdot \log_2 \eta_2 . \qquad (1.1.8)$$

Основной исходный параметр, на котором базируются все расчеты метрических характеристик будущего ПС, – количество имен входных и выходных переменных η_2^*, представленных в предельно краткой записи (с точки зрения алгоритмической сложности – сжатой). Например, для задания одномерного массива (т. е. строки), каково бы ни было число его элементов, требуется всего два имени:

- указатель адреса начала массива;
- количество элементов в нем.

Точно так же для задания двумерного массива достаточно иметь три параметра:

- указатель адреса первого элемента;
- число столбцов;
- число строк.

Если параметр η_2^*, определенный таким образом, рассматривать как размер генеральной совокупности имен входных и выходных переменных, то величина словаря программы η_2 в соответствии с соотношением Холстеда не будет превосходить $\eta_2^* \cdot \log \eta_2^*$. Таким образом, можно считать, что

$$\eta_2 \approx \eta_2^* \cdot \log_2 \eta_2^* . \qquad (1.1.9)$$

Предположим, что существует некоторый язык программирования, называемый *потенциальным*, в котором все программы (по крайней мере – для некоторой предметной области) уже написаны и представлены в виде заранее подготовленных процедур или функций. Тогда для реализации любого алгоритма на таком языке потребуется всего два оператора (функция и присваивание) и η_2^* имен входных и выходных переменных. Это объясняется тем, что в таком потенциальном языке программист должен будет только выбрать нужную процедуру или функцию и применить ее к нужной переменной. Поскольку в такой записи никакие слова не повторяются, то длина программы совпадает с ее объемом и равна:

$$V^* = \left(\eta_2^* + 2\right) \cdot \log_2 \left(\eta_2^* + 2\right). \tag{1.1.10}$$

Эта величина называется *потенциальным объемом* (минимально возможным), соответствующим максимально компактному тексту программы, реализующей данный алгоритм. Это объясняется тем, что в потенциальном языке минимизировано число операторов, а все операнды сведены к перечню процедур или функций и списку входных и выходных переменных.

В таком случае можно определить *уровень реализации* программы, который рассчитывается с помощью отношения

$$L = \frac{V^*}{V}. \tag{1.1.11}$$

Уровень реализации представляет собой метрический показатель, который характеризует степень компактности программы, экономичность использования средств алгоритмического языка. Чем ближе значение L к единице, тем более совершенна программа.

При переводе алгоритма с одного языка на другой его потенциальный объем V^* не изменяется, но действительный объем V может увеличиваться или уменьшаться в зависимости от развитости языков программирования.

Для потенциального языка справедливо равенство $V = V^*$, для любого менее развитого языка следует учитывать соотношение $V > V^*$. Это обусловлено тем, что для потенциального языка $N^* = \eta^*$, в то время как для всех других языков применяется уравнение длины и учет соотношения $N > \eta$.

Оптимизация количества и длины модулей в программе

Соотношение Холстеда позволяет контролировать процесс разработки программных средств, благодаря тому, что если заранее определить их длину, то можно выходить на эту заданную длину модулей при проектировании программ. Действительно, длина модулей (в пределах точности этого соотношения) определяется только их словарями. В свою очередь, величина словарей зависит только от числа групп, на которые разбивается η_2^* входных и выходных переменных программируемой задачи. Таким образом, величина словаря модуля – контролируемый параметр, а если это так, то, следовательно, и его длина может контролироваться.

Пусть k – число модулей. Тогда словарь операндов одного модуля будет определяться следующим выражением:

$$\eta_{2k} = \frac{\eta_2^*}{k} \cdot \log_2 \frac{\eta_2^*}{k}, \tag{1.1.12}$$

а для всей программы (с учетом того, что в программе содержится k модулей) данное выражение будет иметь вид:

$$\eta_2 = \eta_2^* \cdot \log_2 \frac{\eta_2^*}{k}. \tag{1.1.13}$$

Понятно, что каждой группе присваивается имя, которое замещает эту группу в тексте программы. С учетом этого обстоятельства окончательно имеем:

$$\eta_2 = \eta_2^* \cdot \log_2 \frac{\eta_2^*}{k} + k \cdot \log_2 k. \tag{1.1.14}$$

Наилучшее количество модулей, которое будет обеспечивать минимальную длину программы, можно найти следующим образом:

$$k_{\text{опт}} \approx \frac{\eta_2^*}{\log_2 2\eta_2^*}. \tag{1.1.15}$$

При этом число входных имен каждого модуля будет равно:

$$\eta_{2k}^*\Big|_{k_{\text{опт}}} = \log_2 2\eta_2^*. \tag{1.1.16}$$

Исследуя надежность программных средств, Морис Холстед в своих работах показал, что наименьшее количество ошибок обнаруживается в модулях, число входных переменных которых не превосходит восьми, т. е. при $\eta_{2k}^{*} \approx 8$.

Программные средства реальных информационных систем имеют сложную иерархическую структуру. Как правило, самый нижний уровень, на котором располагаются «исполнительные» модули, является наиболее многочисленным, в то время как в верхней части управляющей «пирамиды» размещается один модуль, являющийся головным. Изучая структуры больших программных систем, В.В. Липаев установил, что наиболее жизнеспособными являются такие программные средства, число уровней которых не превосходит 7–8.

Проблема структуризации проектируемых программных средств, безусловно, имеет сугубо содержательный характер и принципиально не может быть формализована. Однако расчетные метрические характеристики (длина модулей, их число, количество иерархических уровней) задают оптимальные параметры структуры программных средств, наиболее рациональные в аспекте обеспечения качества реализации проекта.

Оценка уровня языков программирования

Если N – длина программы, а η – словарь программы, то общее количество выборок необходимых элементов словаря (т. е. фактически – работа программирования) в соответствии с законом Хика будет равна объему программы $N \cdot \log \eta$.

В то же время необходимо еще учесть уровень реализации программы L: количество выборок при этом возрастет в $1/L$ раз. Обозначив работу программирования символом E и учитывая формулу для вычисления уровня реализации программы (1.1.11), окончательно получим:

$$E = \frac{N \cdot \log_2 \eta}{L} = \frac{V}{L} = \frac{V^2}{V^*}. \qquad (1.1.17)$$

В начале 1980-х гг. Холстед ввел формальное определение уровня языка программирования [10], определив этот уровень языка следующим образом:

$$\lambda = L \cdot V^*, \qquad (1.1.18)$$

где L – уровень реализации программы, V^* – ее потенциальный объем.

Для любого алгоритма, который программируется с использованием разных языков, с увеличением объема уровень реализации уменьшается в той же пропорции. В результате произведение уровня L на объем V равняется потенциальному объему V^* данного алгоритма. С другой стороны, если язык реализации остается одним и тем же, а разрешено менять сам алгоритм, имеется другое, но похожее соотношение. В этом случае с увеличением потенциального объема V^* уровень программы L уменьшится в том же отношении. Следовательно, произведение L на V^* остается неизменным для любого языка.

Определим работу программирования при использовании потенциального языка. В записи на потенциальном языке программа имеет минимально возможную длину, и так как слова (операнды и операторы) в ней не повторяются, то она совпадает с объемом:

$$V^* = \left(\eta_2^* + 2\right)\log_2\left(\eta_2^* + 2\right). \tag{1.1.19}$$

Работа программирования в потенциальном языке сводится к выбору из конечного, но огромного по масштабам числа имен функций и процедур:

$$2^1 + 2^2 + 2^3 + \ldots + 2^{V-1} + 2^V \approx 2^{V+1}. \tag{1.1.20}$$

где V – объем программы.

Существует закон Хика, в соответствии с которым время реакции при выборе из некоего числа альтернативных сигналов зависит от их количества. Тогда по закону Хика работа выбора из библиотеки функций составит $\log 2^{V+1} \approx V$, и полная работа программирования при использовании потенциального языка будет определяться следующим образом:

$$\mathrm{E}_{\text{пот}} = V^* \cdot V. \tag{1.1.21}$$

В таком случае мы получим следующее соотношение:

$$\lambda = \frac{E_{\text{пот}}}{E} = V^* \cdot V : \frac{V^2}{V^*} = \frac{V^{*2}}{V}. \tag{1.1.22}$$

Полученное значение λ как раз можно считать количественной мерой уровня любого алгоритмического языка.

Из выражения (1.1.22) видно, что для постоянства λ увеличение объема программы должно квадратично зависеть от увеличения объ-

ема информации по внешним связям. Поэтому алгоритмически сложные программы вычисления малого числа переменных будут давать значительно более низкое значение λ, чем программы вычисления большого числа переменных по элементарным выражениям.

В связи с этим метрику уровня языка программирования λ для сравнения языков следует применять только для конкретной предметной области и близких типов задач. Ниже приведены данные об уровнях некоторых известных языков программирования (табл. 1.1).

Таблица 1.1. Уровни языков программирования

Язык	λ	Отклонения
Естественный язык	2,16	0,74
PL/1	1,53	0,92
Алгол	1,21	0,74
Паскаль	1,25	0,76
Бейсик	1,22	0,72
Фортран	1,14	0,81
Ассемблер	0,88	0,42

Поскольку известны соотношения для работы программирования E и уровня языка λ

$$E = \frac{V^2}{V^*} \text{ и } \lambda = \frac{V^{*2}}{V}, \tag{1.1.23}$$

то, исключая из выражений величину V, получим:

$$E = \frac{V^{*3}}{\lambda^2}. \tag{1.1.24}$$

Тогда квалификационное время программирования будет определяться следующим образом:

$$T = \frac{E}{S} = \frac{V^{*3}}{S\lambda^2}, \tag{1.1.25}$$

где S – число Страуда ($5 \le S \le 20$, среднее его значение принято считать равным 18).