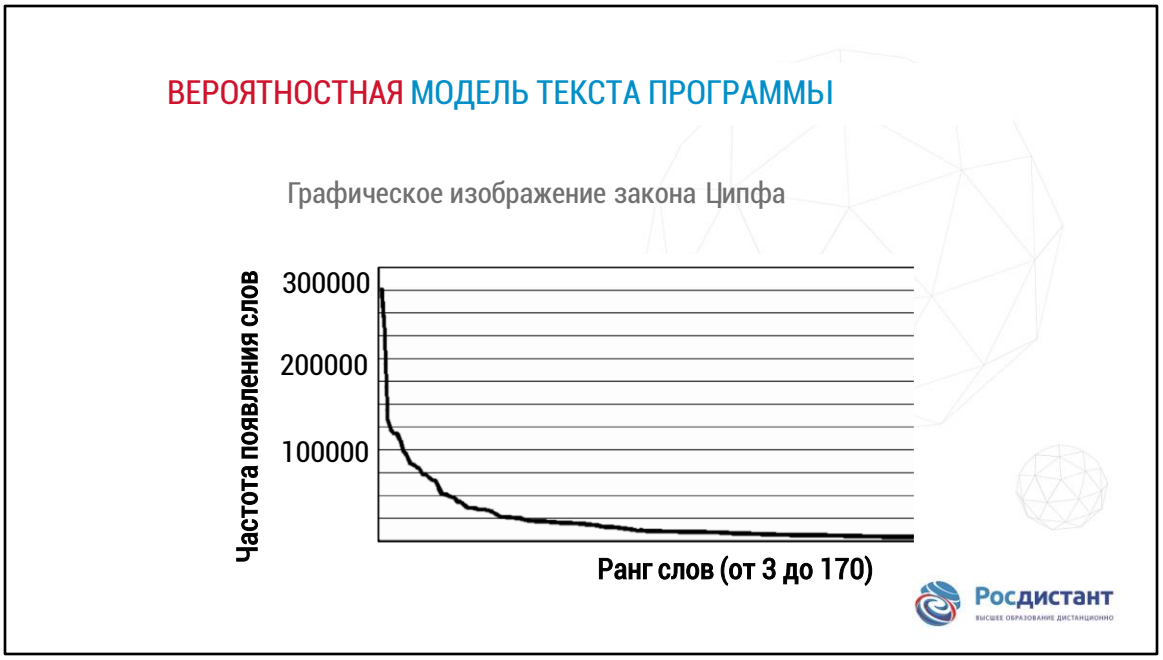


Тема 3.2. Лексический анализ программ

Рассмотрим оценку характеристик программ на основе лексического анализа.

Лингвистические исследования количественных характеристик текстов позволили установить ряд эмпирических закономерностей, имеющих важное значение и для информационных технологий. «Одной из таких закономерностей является закон Ци́пфа.

Этот закон определяет зависимость между частотой появления тех или иных слов в тексте и его длиной» [14].

Американский лингвист Джордж Ципф из Гарвардского университета в 1949 году установил эмпирическую закономерность распределения частоты слов естественного языка.

Если все слова длинного текста упорядочить по убыванию частоты их использования, то частота *n*-го [энного] слова в таком списке окажется приблизительно обратно пропорциональной его порядковому номеру *n* [эн]. Этот порядковый номер называется рангом слова.

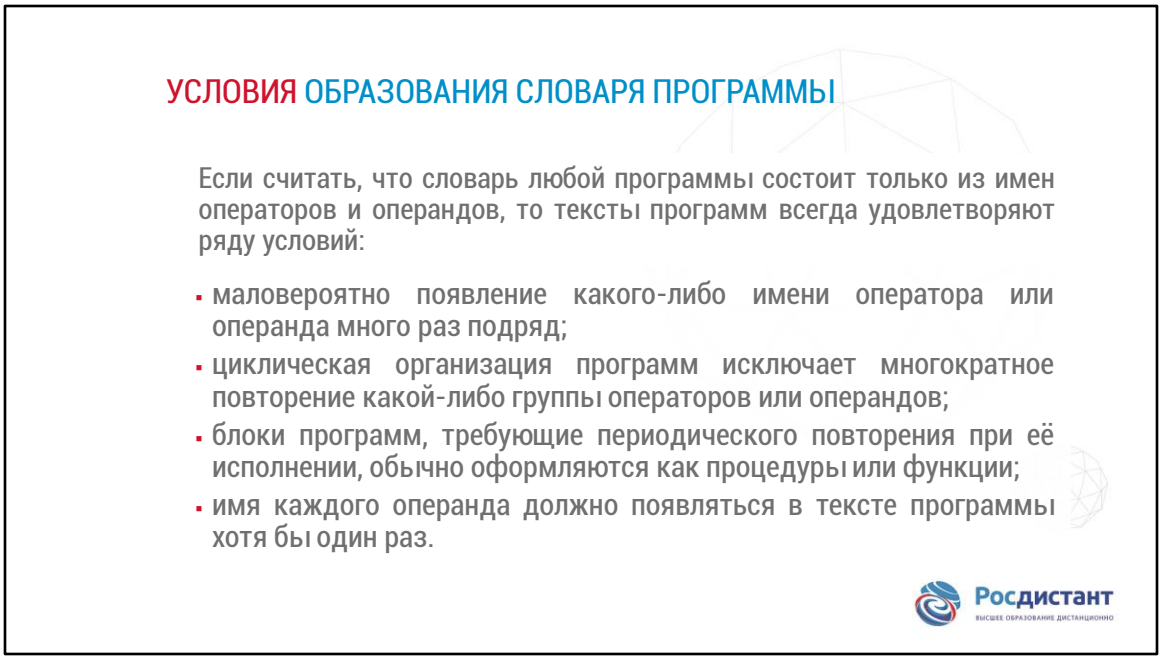
В отношении используемого языка это означает следующее:

– существует небольшое количество слов, которые используются чрезвычайно часто: левая часть графика;

– есть достаточно большое количество слов, которые используются достаточно часто: средняя часть графика;
– очень большое количество слов практически никогда не используются: правая нижняя часть графика.

«Аналогичные результаты были получены Холстедом в 1977 году при изучении текстов программ, написанных на различных алгоритмических языках. Кроме того, эмпирически им было найдено соотношение, связывающее общее количество слов в программах с величиной их словарей.

Как оказалось, впоследствии эти закономерности могут быть получены и объяснены теоретически на основе представлений алгоритмической теории сложности» [14].



Программа определяет последовательность операторов, которые выполняют действия над опера́ндами. «Операнд – это некоторый объект или величина, обрабатываемая в программе, а оператор представляет собой обозначение действия, выполняемого по отношению к операнду» [14].

Если считать, что словарь программы состоит только из имен операторов и операндов, то текст

программы должен удовлетворять 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 достаточно развитых возможностей языков программирования;
- блоки программ, требующие периодического повторения при исполнении программы, 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. Неиспользуемые имена следует удалять из текста программ, чтобы сократить объем памяти, используемой при объявлении переменных» [14].

Каждая программа может рассматриваться как результат сжатия ее развёртки за счет применения циклов, функций и процедур.

В соответствии с теорией алгоритмической сложности «длина программы представляет собой меру сложности развёртки. Ее текст не может быть значительно сокращен, поскольку все закономерности развёртки уже использованы для этой цели» [14].

ИЗМЕРЯЕМЫЕ СВОЙСТВА ПРОГРАММ

В состав измеримых свойств программы могут быть включены следующие метрические характеристики:

η_1 – число простых операторов, появляющихся в данной реализации;

η_2 – число простых 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.

В конкретной реализации текста программы можно определить:

- величину словаря $\eta = \eta_1 + \eta_2$; (5)
- длину реализации программы $N = N_1 + N_2$; (6)
- длину программы $\tilde{N} = (\eta_1 \cdot \log_2 \eta_1) + (\eta_2 \cdot \log_2 \eta_2)$; (7)
- объем программы $V = \mathrm{N} \cdot \log_2 \eta = \eta \cdot \log_2^2 \eta$. (8)

Соотношение Холстеда:

$$N = \eta_1 \cdot \log_2 \eta + \eta_2 \cdot \log_2 \eta \approx \eta_1 \cdot \log_2 \eta_1 + \eta_2 \cdot \log_2 \eta_2 = N_1 + N_2. \quad (9)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В исходном тексте программы можно идентифицировать все операнды, определенные как переменные или константы, и операторы, определенные как последовательности символов.

В состав измеряемых свойств программы могут быть включены следующие метрические характеристики:

– число простых операторов и операндов,

появляющихся в данной реализации;

– общее число всех операторов и операндов, появляющихся в данной реализации.

Учитывая вышеперечисленные характеристики, можно рассчитать по формулам 5–8 такие метрические характеристики, как размер словаря, длина реализации программы, длина программы, объем программы.

«Словарь программы состоит только из операторов и операндов. Длина реализации и объем программы определяются на основе анализа текста программы путем подсчета количества операторов и операндов. Также учитывается число их вхождений в текст программы» [14]. Иными словами, длина реализации и объем программы определяются на основе лексического анализа.

«Длина программы представляет собой математическое ожидание количества слов в тексте

программы. В отличие от длины программы объем измеряется не количеством слов, а числом двоичных разрядов» [14].

Существует взаимная связь между длиной программы и размером словаря, а также между величиной словаря и объемом программы. По известному размеру словаря можно найти значения длины реализации программы и объем программы. Соотношение между величиной словаря и длиной текста единственно и взаимно однозначно. Учитывая принятые обозначения, получим соотношение Холстеда по формуле 9.

«Метрики длины программы и длины реализации так же могут использоваться для выявления несовершенств программирования, возникших вследствие применения не очень удачных приемов программирования» [14].

УРОВЕНЬ РЕАЛИЗАЦИИ ПРОГРАММЫ

Потенциальный объем программы:

$$V^* = (\eta_2^* + 2) \cdot \log_2(\eta_2^* + 2), \quad (10)$$

где η_2^* – число имен входных и выходных переменных.

Уровень реализации программы:

$$L = \frac{V^*}{V}, \quad (11)$$

где V – действительный объем программы.

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 $N > \eta$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

«Пусть существует некоторый язык программирования, называемый потенциальным, в котором все программы 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. А также число имен входных и выходных переменных.

В таком потенциальном языке программист должен будет только выбрать нужную процедуру или функцию и применить ее к конкретной переменной.

Поскольку в такой записи никакие слова не повторяются, то длина программы совпадает с ее объемом.

Такая величина называется потенциальным объемом, соответствующим максимально компактному тексту программы, реализующей данный алгоритм» [1]. Потенциальный объем программы определяется по формуле 10.

Таким образом, в потенциальном языке минимизировано число операторов, а все операнды сведены к перечню процедур или функций и списку входных и выходных переменных.

По формуле 11 можно рассчитать уровень реализации программы. Этот метрический показатель характеризует степень компактности программы,

экономичность использования средств алгоритмического языка.

Чем ближе значение переменной L [эль] к единице, тем более совершенна программа.

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

ОПТИМИЗАЦИЯ КОЛИЧЕСТВА И ДЛИНЫ МОДУЛЕЙ В ПРОГРАММЕ

Наилучшее количество модулей, которое будет обеспечивать минимальную длину программы:

$$k_{\text{опт}} \approx \frac{\eta_2^*}{\log_2 2\eta_2^*}. \quad (12)$$

Число входных имен каждого модуля:

$$\eta_{2k}^* \mid_{k_{\text{опт}}} = \log_2 2\eta_2^*. \quad (13)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Процесс разработки программных средств позволяет контролировать соотношение Холстеда. Если заранее определить длину программы, то можно выходить на заданную длину модулей при проектировании программ.

Длина модулей определяется только размером их словарей, который зависит только от числа групп, на которые разбивается число входных и выходных

переменных программы.

Таким образом, величина словаря модуля – контролируемый параметр. Следовательно, и его длина может контролироваться.

«Исследуя надежность программных средств, Мори́с Хо́лстед в своих работах показал, что наименьшее количество ошибок обнаруживается в модулях, число входных переменных которых не превосходит восьми» [1].

Программные средства реальных информационных систем имеют сложную иерархическую структуру. Самый нижний уровень, на котором располагаются исполнительные модули, является наиболее многочисленным. В верхней же части размещается один модуль, являющийся головным.

Изучая структуры программных систем, Липа́ев Владимир Васильевич – один из основателей

отечественной школы программной инженерии – установил, что наиболее жизнеспособными являются программные средства, число уровней которых не более 7-8.

«Проблема структуризации проектируемых программных средств имеет содержательный характер и принципиально не может быть формализована.

Однако расчетные метрические характеристики, такие как длина и число модулей, количество иерархических уровней, задают оптимальные параметры структуры программных средств, наиболее рациональные в аспекте обеспечения качества реализации проекта» [1].

Количество модулей, обеспечивающее минимальную длину программы, и число входных имен каждого модуля определяются соотношениями 12 и 13 соответственно.

КОЛИЧЕСТВЕННАЯ ОЦЕНКА РАБОТЫ ПРОГРАММИРОВАНИЯ

Закон Хика: время реакции выбора из альтернативных действий на равновероятные раздражители логарифмически связано с числом альтернатив.

Если N – длина программы, а η – словарь программы, то общее количество выборок в соответствии с законом Хика будет равно объему программы $N \cdot \log \eta$.

Обозначив работу программирования символом E и учитывая формулу для вычисления уровня реализации программы (9), получим:

$$E = \frac{N \cdot \log_2 \eta}{L} = \frac{V}{L} = \frac{V^2}{V^*}. \quad (14)$$

Время программирования (квалификационное время) вычисляется по формуле: $T = \frac{E}{S} = \frac{V^2}{SV^*}$, (15)

где S – число Страуда ($5 \leq S \leq 20$, среднее его значение равно 18).

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В 1952 году английский психолог Ви́льям Э́дмунд Хик вывел закон, в котором говорится, что время реакции при выборе из некоторого числа альтернативных сигналов зависит от их количества.

Число сравнений при формировании нужной выборки может служить объективной мерой работы программирования в вероятностной модели программы. Следовательно, оно справедливо для

оценки процесса программирования.

Уравнение 14 показывает, что мысленная работа по реализации любого алгоритма с данным потенциальным объемом в каждом языке пропорциональна квадрату объема программы.

Правильное разбиение программы на модули может уменьшить работу по программированию за счет сокращения общего объема выборок по отдельным частям программы.

Джон Стра́уд в работе «Тонкая структура психологического времени» определил «момент» как время, требуемое человеческому мозгу для выполнения наиболее элементарного различения. Число Страуда, то есть число мысленных сравнений, производимых человеком в единицу времени, лежит в пределах от 5 до 20. Среднее его значение принято считать равным 18 [восемнадцати].

Квалификационное время программирования

рассчитывается по формуле 15 с учетом соотношения 14, приведенного выше.

Формула 15 для учета календарного времени разработки программного средства не используется.

Оценка реального календарного времени разработки нового программного средства основана на статистике производительности труда программистов. При этом часто используется расчет по аналогам, когда продолжительность разработки определяют по схожим условиям и объемам проектирования.

ОЦЕНКА УРОВНЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Формула Холстеда:

$$\lambda = L \cdot V^* = \frac{(V^*)^2}{V}, \quad (16)$$

где λ - уровень языка программирования; $L = \frac{V^*}{V}$ – уровень реализации программы.

Уровни языков программирования:
Естественный язык $\lambda = 2{,}16$; Паскаль $\lambda = 1{,}25$; Бейсик $\lambda = 1{,}22$; Ассемблер $\lambda = 0{,}88$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

В 1980-х годах Холстед ввел «формальное определение уровня языка программирования с помощью соотношения 16.

Для любого алгоритма, который программируется с использованием разных языков, с увеличением объема уровень реализации уменьшается в той же пропорции. В результате произведение уровня *L* [эль] и объема *V* [вэ]

равняется потенциальному объему 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 уровень программы уменьшится в том же отношении.

Следовательно, произведение уровня реализации программы и ее потенциального объема остается неизменным для любого языка» [1].

Статистические исследования показали, что величина λ [лямбда] зависит только от конкретного языка программирования и практически не зависит от объема программы.

Величина λ [лямбда] считается постоянной в пределах одного и того же языка.

«Метрику уровня языка программирования λ [лямбда] для сравнения языков следует применять

только для конкретной предметной области и близких типов программ.

В таблице приведены данные об уровнях некоторых известных языков программирования. Как видно, значения уровня языка заключены в довольно узком диапазоне» [1].

МЕТРИКА ЧИСЛА ОШИБОК В ПРОГРАММЕ

Количество ошибок в программе определяется по формуле:

$$B = L\frac{E}{E_0} = \frac{V}{E_0} = \frac{(V^*)^2}{E_0\lambda}, \quad (17)$$

где L – уровень реализации программы;

$V = \frac{(V^*)^2}{\lambda}$ – объем программы;

E – число элементарных мысленных различений;

E_0 – среднее число элементарных различений между возможными ошибками в программировании.

Если предположить, что с каждым из фрагментов программы связана, по крайней мере одна ошибка, то получим:

$$B_0 = \frac{E}{E_0} = \frac{V}{3000}. \quad (18)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Значительная часть усилий и времени, которые затрачиваются на создание большинства программных продуктов, приходится на их отладку. Под отладкой понимается выявление и устранение ошибок, внесенных в начальный период написания программы.

Следовательно, любое обоснованное представление о количестве первоначальных ошибок,

ожидаемых в данной программе, даст важную оценку для практики.

Рассматриваемая метрика позволяет предсказать число первоначальных ошибок до отладки и тестирования. Однако она не может служить свидетельством правильности программы, даже если ее значение равно нулю.

Время, требуемое на разработку программы, характеризуется числом элементарных мысленных различений E [е]. Следовательно, число моментов, в которые можно сделать ошибочное различение, также определяется значением E [е] или связанным с ним значением объема программы V [вэ].

На количество ошибок в программе будет влиять и сам язык программирования, который можно учесть при помощи уровня реализации программы: формула 17.

При вычислении ошибок в программе следует

помнить, что для определения потенциального объема необходимо только знание числа независимых входных и выходных параметров программы. Это число задается в техническом задании на разработку программного средства.

После выбора языка программирования потенциальное количество ошибок можно оценить до начала написания программы, то есть до начала проектирования.

Опытным путем установлено, что количество ошибок в текстах программ пропорционально работе программирования, которая может быть вычислена, как указано выше.

Кроме того, формирование текста программы происходит не как цельный готовый продукт, а в виде некоторых фрагментов ограниченного объема. Причем размеры этих фрагментов зависят от очень многих факторов. Если предположить, что с каждым

из таких фрагментов связана хотя бы одна ошибка, то получим соотношение 18.

ПОРЯДОК РАСЧЕТА МЕТРИЧЕСКИХ ХАРАКТЕРИСТИК

1. Расчет структурных параметров ПС. Число модулей ПС:

$$k = \frac{\eta_2^*}{8}. \quad (19)$$

Если $k \gg 8$, то структура ПС будет многоуровневой. Число уровней:

$$i = \left[\frac{\log_2 \eta_2^*}{3}\right] + 1. \quad (20)$$

В этом случае необходима поправка на число модулей:

$$K = \frac{\eta_2^*}{8} + \frac{\eta_2^*}{8^2} + \cdots + \frac{\eta_2^*}{i}. \quad (21)$$

2. Расчет длины программы. Полная длина ПС равна

$$N = KN_K + K\log_2 K = 220K + K\log_2 K\,. \quad (22)$$

3. Расчет объема ПС. Объем одного модуля:

$$V_k = N_k \log_2 2\eta_{2k}. \quad (23)$$

Общий объем программы, содержащей K модулей:

$$V \approx K \cdot V_k = K \cdot N_k \log_2 2\eta_{2k}\,. \quad (24)$$

4. Расчет количества команд ассемблера.

$$P = \frac{3}{8}N, \quad (25)$$

где 3/8 – коэффициент пересчета Кнута на команды ассемблера.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Анализ структуры входных и выходных данных – наиболее ответственный момент в процессе проектирования программных средств. Требуется точность вычисления метрических характеристик программного средства, которая может быть достигнута в соответствии с соотношением Холстеда. Она может быть получена при правильной оценке количества имен входных и выходных переменных.

Рассмотрим один из вариантов этого расчета.

Первый шаг – расчет структурных параметров программного средства. Формулы 19–21.

Приняв рекомендацию Холстеда об оптимальном количестве входных переменных модуля, которое должно быть равно восьми, находим число модулей программного средства по формуле 19. Число уровней иерархии на практике обычно не превышает восьми.

Второй шаг – расчет длины программы, формула 22.

Поскольку в данном выражении уровни первого и второго членов существенно различаются, последним слагаемым часто пренебрегают, ограничиваясь доминирующим значением первого.

Третий шаг – расчет объема программного средства.

Сначала вычисляется объем одного модуля по

формуле 23. Общий объем программы, содержащий K [ка] модулей, определяется по формуле 24 как кратное найденной величины количеству модулей. Значение вычисляется приблизительно, поскольку модули могут иметь различную длину.

Четвертый шаг – расчет количества команд ассе́мблера.

Если на основании постановки задачи известно, что проектируемое программное средство включает N [эн] слов, то количество команд ассемблера определяется по формуле 25. Формула содержит коэффициент пересчета Кну́та на машинные команды.

ПОРЯДОК РАСЧЕТА МЕТРИЧЕСКИХ ХАРАКТЕРИСТИК

5. Расчет календарного времени программирования.

$$T_k = \frac{P}{n \cdot \nu} = \frac{3 \cdot N}{8 \cdot n \cdot \nu}, \quad (26)$$

где n – количество программистов в бригаде разработчиков; ν – производительность (число отлаженных команд в день), устанавливаемая директивно в пределах $10 \leq \nu \leq 30$.

6. Расчет начального количества ошибок (перед комплексной отладкой).

$$B_0 = \frac{V}{3000}, \quad (27)$$

где V – расчетный объем ПС.

7. Расчет начальной надежности программного средства.

Период отладки τ в пределах календарного времени разработки T_k: $\frac{1}{2}T_k \leq \tau \leq \frac{2}{3}T_k$. Пусть время отладки занимает у разработчиков половину рабочего времени: $\tau = \frac{1}{2}T_k$. Тогда время наработки программы на отказ:

$$t_n = \frac{\tau}{\ln B_0} = \frac{T_k}{2 \ln B_0}. \quad (28)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Рассмотрим последующие действия по расчету метрических характеристик программных средств.

Пятый шаг – расчет календарного времени программирования, формула 26. Оценка календарного времени программирования основана на производительности труда программистов. Установлено, что при пересчете на команды ассемблера производительность программистов

составляет от 10 до 30 отлаженных команд за рабочий день. Рабочий день примем за семь часов.

В каждом коллективе программистов величина дневной выработки устанавливается, исходя из реальных экономических условий, однако часто – директивным образом.

Шестой шаг – расчет начального количества ошибок перед комплексной отладкой по формуле 27. Как отмечалось ранее, текст программы формируется не целиком, а фрагментами ограниченного объема. Причем размеры этих фрагментов зависят от множества факторов. Предполагается, что с каждым из таких фрагментов связана по крайней мере одна ошибка.

Седьмой шаг – расчет начальной надежности программного средства. Отладка различных программных средств может занимать различное время в зависимости от сложности программы.

Отладка занимает бóльшую часть этапа проектирования программ.

Предположим, что время отладки занимает у разработчиков половину рабочего времени. Тогда время наработки программы на отказ, то есть работы программного средства до появления первого сбоя, определяется по формуле 28.

Как следует из приведенной последовательности расчетов, такой подход с использованием основ метрической теории программ позволяет управлять разработкой программного средства по следующим направлениям:

– по срокам: благодаря изменению количества программистов, зáнятых в проектировании;
– по начальной надежности: за счет изменения доли времени отладки в пределах календарного времени разработки или увеличения самогó периода разработки.

Данный подход способствует оптимизации объемных и структурных параметров программного средства. Кроме того, он обеспечивает расчет себестоимости всей разработки программ на основе оценки трудоемкости.

МЕТРИКИ ЧЕПИНА (NED CHAPIN)

Множество переменных, составляющих список ввода-вывода, разбивается на следующие группы:
P – вводимые переменные для расчетов и для обеспечения вывода;
M – модифицируемые, или создаваемые внутри программы, переменные;
C – переменные, участвующие в управлении работой программного модуля;
T – не используемые в программе переменные.

Выражение для определения метрики Чепина в общем виде:

$$Q = a_1 * P + a_2 * M + a_3 * C + a_4 * T, \quad (29)$$

где a_1, a_2, a_3, a_4 – весовые коэффициенты.

С учетом весовых коэффициентов расчетное выражение метрики Чепина приобретает следующий вид:

$$Q = P + 2 \cdot M + 3 \cdot C + 0{,}5 \cdot T. \quad (30)$$

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Метрика Чепина является «мерой сложности понимания программ на основе входных и выходных данных. Смысл метода состоит в оценке информационной прочности отдельно взятого программного модуля. Оценка производится на основе результатов анализа характера использования переменных, входящих в состав списка ввода и вывода» [1].

Имеется несколько вариантов метрики Чепина. Рассмотрим более эффективный вариант.

«Всё множество переменных, составляющих список ввода-вывода, разбивается на четыре функциональные группы:

– вводимые переменные для расчетов и для обеспечения вывода. Это может быть, например, используемая в программах лексического анализатора переменная, содержащая строку исходного текста программы. В этом случае сама переменная не модифицируется, а применяется только как контейнер для исходной информации;
– модифицируемые, или создаваемые внутри программы, переменные. К таким переменным относится большинство традиционно применяемых переменных, декларируемых в программных модулях;
– переменные, участвующие в управлении работой

программного модуля. Такие переменные предназначены для передачи управления, изменения логики вычислительных процессов;

– не используемые в программе переменные. Такие переменные не принимают непосредственного участия в реализации процесса обработки информации, ради которого написана анализируемая программа. Однако они заявлены в программном модуле» [1].

Применение данной метрики имеет свою особенность. Так, необходим учет каждой переменной в каждой функциональной группе, поскольку каждая переменная может выполнять одновременно несколько функций.

Исходное выражение для определения метрики Чепина записывается в виде формулы 29, а с учетом весовых коэффициентов – в виде формулы 30.

МЕТРИКИ ДЖИЛБА (THOMAS JILB)

Показатели программного средства:
• CL – абсолютная сложность программы, характеризуемая количеством операторов условий;
• cl – относительная сложность программы, определяющая насыщенность программы операторами условия, равна отношению CL к общему числу операторов L.

Характеристики программы:
• количество операторов цикла L_{loop};
• количество операторов условия L_{IF};
• число модулей или подсистем L_{mod};
• отношение числа связей между модулями к числу модулей: $f = \frac{N_{SV}^4}{L_{mod}}$; (31)
• отношение числа ненормальных выходов из множества операторов к общему числу операторов: $f^* = \frac{N_{SV}^*}{L}$. (32)

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Для оценки качества программного обеспечения достаточно практичными являются метрики, предложенные То́масом Джи́лбом. Его метрики основаны на результатах анализа текстов программных продуктов.

В качестве меры логической трудности Джилб предложил число логических «двоичных принятий решений». Логическую сложность программы

определяет насыщенность программы условными операторами и операторами цикла.

Джилб считал нужным применять следующие характеристики программы:

- «количество операторов цикла;
- количество операторов условия;
- число модулей;
- отношение числа связей между модулями к числу модулей, формула 31;
- отношение числа необычных выходов из операторов, в которых происходит принятие решений, к общему числу операторов, формула 32» [1].

Среди всех показателей качества программ Джилб указывает надежность программы. Ее он характеризует как возможность того, что данная программа проработает определенное время без логических сбоев.

Программную надежность автор предлагает рассчитывать так: «единица минус отношение числа логических сбоев к общему числу запусков» [1].

Отношение количества правильных данных ко всей совокупности данных приводится в качестве меры точности – свободы от ошибок. Точность нужна как средство обеспечения надежности программы.

Прецизионность определяется как мера того, насколько часто появляются ошибки, вызванные одинаковыми причинами. Автор оценивает этот показатель дробью. В числителе указывается количество фактических ошибок на входе. В знаменателе – общее число ошибок, причинами которых явились эти ошибки на входе.

Так, например, если одна ошибка вызывает в течение определенного времени появление 50 сообщений об ошибках, то прецизионность составляет 0,02 [ноль целых две сотых].

ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММЫ НА ОСНОВЕ ЛЕКСИЧЕСКОГО АНАЛИЗА

Задача. Необходимо разработать программу для вычисления значений функции:

$$f(x) = \begin{cases} 0{,}5 \text{ при } x \le 0{,}5; \\ x + 1 \text{ при } -0{,}5 < x \le 0; \\ x^2 - 1 \text{ при } 0 < x \le 1; \\ x - 1 \text{ при } x > 1. \end{cases}$$

Оценить её качество с использованием метрик Джилба.
Написать текст программы на C#.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

Рассмотрим пример оценки характеристик программы на основе лексического анализа.

Необходимо разработать программу для вычисления значений функции и на основе лексического анализа исходного текста программы оценить ее качество с использованием метрик Джилба.

Программа для реализации алгоритма

вычисления значений функции разработана на языке программирования C# [си шарп].

В программе не используются операторы цикла. Повторение процедур выполнения операторов программы реализовано с помощью оператора goto [го́у ту]. Его применение не рекомендуется при разработке профессионального ПО, поскольку затрудняет понимание программы.

В учебных целях применение данного оператора будем считать допустимым. Такой оператор не влияет на параметры оценки качества программы с применением метрик Джилба.

Программа, разработанная для реализации заданного алгоритма, не имеет циклов и модулей. Следовательно, из перечисленного набора характеристик можно определить только:

- уровень реализации программы;
- количество операторов условия;

– абсолютную сложность программы;
– относительную сложность программы, которая вычисляется как отношение абсолютной сложности к общему числу операторов.

При этом абсолютная сложность программы равна количеству операторов условия.

ПРИМЕР ОЦЕНКИ ХАРАКТЕРИСТИК ПРОГРАММЫ НА ОСНОВЕ ЛЕКСИЧЕСКОГО АНАЛИЗА

Составить словарь операторов и операций программы в виде таблицы:

№	Операторы, операции	Номер строки
1	using …	1
2	class …	3
3	{}	4 (35), 6 (34)
4	public static void	5
5	float	7
6	char	9
7	string	10
8	Console.Clear()	13
	…	…

Определить: L_{IF}, CL, $cl = CL / L$.

Росдистант
ВЫСШЕЕ ОБРАЗОВАНИЕ ДИСТАНЦИОННО

При подсчете общего количества операторов программы будем руководствоваться правилами, определенными при подсчете операторов в метриках Холстеда.

В таблице на слайде приведены операторы и операции, используемые в программе. При подсчете числа операторов следует иметь в виду, что в двадцать восьмой строке текста программы скобки

используются как символы стро́ковых констант, поэтому операторами не являются.

Согласно теории Джилба, необходимо определить количество операторов условия, используемых в исходном тексте программы для реализации алгоритма решения поставленной задачи.

В языке программирования C# [си шарп] к такой категории инструкций относятся условные операторы ветвления и операторы циклов. В своем составе они содержат логическое выражение, являющееся условием выполнения указанных операторов.

В зависимости от соблюдения условия программа может существенно изменить ход выполнения. Поэтому значимость условных операторов достаточно велика.

В исходном тексте программы используются условные операторы, которые расположены в строках

с номерами 17, 18, 23 и 33.

Получаем следующие результаты оценки характеристик программы на основе метрик Джилба:

- число условных операторов равно четырем;
- абсолютная сложность программы равна четырем, так как в программе используются четыре оператора условия;
- относительная сложность программы составляет 0,0435 [ноль целых четыреста тридцать пять десятитысячных].

С точки зрения лексического анализа исходного текста программы представленное решение не является сложным. Количество ветвлений в программе невелико, четыре оператора условия, что подтверждается невысокой величиной метрики относительной сложности программы.