UDC 004.412:519.876.5

V.O. MISHCHENKO

V.N. Karazin Kharkov National University, Ukraine

DOES THE DIFFERENT DEFINITIONS OF ADA PROGRAM TOKENS HAVE SIGNIFICANT DIFFERENCE?

M. Halstead's Software Science is the origin of some program metrics. Among them are well-known as well as new measures. Their evaluation is commonly used a counting strategy, i.e. a definition of Halstead's tokens of a programming language. We study the differences between the two strategies for Ada programming language, one base on compiler oriented approach (1987) and the other base on human oriented approach (2007). Analytical consideration and testing both shows the first strategy gives higher values of program vocabulary (excess is usually moderate). It is more difficult to compare the respective estimations of program length. Under certain conditions, they approximately equal. Despite the different approaches, the recent counting strategy for Ada 95 (2004-07) seems sufficiently coherent to the previously tested strategy for Ada 83 (1987).

program metrics, Software Science, Ada, tokens, counting strategy, program vocabulary, program length

Introduction

Software science measures [1] have well known since the publication of works Maurice Halstead [2-3]. Recently, the author of the article proposed to modernize some of these metrics and introduce new ones [4-6]. Calculation of metrics, according to the origins of software science, generally requires knowledge of the program (module) vocabulary and length. We can compute these primitives by counting operators and operands that are special tokens of programming language. According Halstead idea, the tokens are graphic images of semantic elements, which combines programmer.

Interest in the use of metrics discussed above automatically makes the task of setting the counting strategies for tokens of modern programming languages. We considered this to Ada 95 language and defined the counting strategy, which satisfy the general principles.

The purpose of this article is a clarification of our counting strategy of consistency with the earlier definition of a strategy for Ada 83. In particular, it is useful to know under what conditions the two strategies can be seen as mutually substitutable for the Ada programs, which satisfy the standard from 1983.

Short history of problem

The counting strategy for the full Ada 83 was published by D. Miller, R. Maness, J. Howatt, and W. Shaw [7]. These authors tested initially two heuristic approaches, which proved unsatisfactory. Only third, based on the BNF description of Ada, had led them to the satisfactory counting strategy [7]. The authors argued the following statement. If any investigator expects to obtain useful estimates, he must count tokens of all program including declarations as well as executable code [7].

Note that one of essential problems of the work [7], that is which tokens are operators, is not important to us because our energy approach avoids it [4].

On the other hand, the "biggest problem" in [7] of classification multi-token language constructions is relevant in any attempt to define a counting strategy. Works [7] and [5-6] both use the standard description language syntax for Ada to solve the problem. But they do so in different ways according to different concepts.

It is remarkable that authors of [7] not interested for psychological complexity of Ada programs. Instead of this, they investigated the utility of software science measures in estimating the resources needed by Ada compilers to translate Ada programs. As a result, they have identified the suitable counting strategy by using the same syntax charts that developers have used in designing Ada compilers. Hence we can say that counting strategy of [7] is compiler oriented.

Defining counting strategy for Ada 95, works [5-6] originated from the submissions more traditional for software science [4, 6]. Correct implementation of one of those rules requires careful matching tokens with the formal syntax and some of the contextual rules [6]. Nevertheless, this approach should be considered as human-oriented one.

Our tasks

To achieve the goal specified in the introduction should perform the following tasks:

− nalyze for each kind tokens from the [7], how it agreed with the definition of tokens from [5],

− predict the nature of the probable evolution of the program vocabulary value (and the length of the program) when you change one strategy to another,

− ascertain the conditions under which the results of measurements on the two strategies should not vary significantly,

− verify the theoretical conclusions with examples of Ada programs.

Analytical Considerations

The Ada 95 language is a strict extension of the Ada 83 language. That’s why we expect that the program under consideration be subject to the standard 1983. We suspect that the calculations of software primitives to be made separately for each of the compilation units of the program.

To carry out our first task, we will consider the differences between the two strategies in order of the list of rules as defined in the Appendix A of the article [7].

1. Comments are not considered in the compiler-oriented strategy. It almost does not change the vocabulary, but could appreciably reduce the length.

2. The second group of rules of the compiler-oriented strategy provides important distinguishing feature. Local variables with the same name in different program modules (loops, blocks) are counted as unique tokens. This leads to an increase in the size of the vocabulary, without affecting the length of the program.

3. The third rule lists 20 the reserved word combinations that they counted as multi-word tokens. Half of them have the same status as in the use of human-oriented strategy. In seven cases, the word combinations have close interpretation in both counting strategies:

array of *do end* *for in loop end loop*
while loop end loop *body is* *exception when*
case when end case (1)

Only three combinations,

and then *or else* *limited private*, (2)

are not included in human-oriented strategy. Reserve words that form them are separate tokens. Change strategy (compiler-oriented instead of human-oriented) would lead to such changes. Vocabulary little change. The length of the program would increase for two of combinations (1)-(2). For the six combinations it will decrease. In other cases, the length will not change.

4. In this group, there are nine rules. Of these, the third, fifth, sixth, seventh rules lead to increase the size of the dictionary. The ninth rule increases the length of the program unit (adding 1), but only for subunits. The third rule is most remarkable. It establishes that parentheses can play the role of eight different program tokens, depending on the context.

5. Tokens of this group are reserved words and delimiters. Their interpretations by the two strategies have been the same or close in 60 cases out of 62. The exceptions are the words "is" and "end", which are considered by the human-oriented strategy only in combinations of reserved words. When rules of this group are applicable, the program vocabulary and length obtain almost equal increment in the use of any of the two strategies.

6. This rule does not affect the counting of the length and vocabulary.

7. Each distinctly declared module identifier is counted as a separate program token. Therefore, unlike the human-oriented strategy, all overloaded declarations of the same subprogram identifier are considered to generate different tokens. Similarly, a type declaration is counted either as one or another kind of tokens depending on its use. It is considered of one kind in its own declaration, but it is considered as token of different kind when it types a variable (function, subtype). This approach increases the size of the vocabulary, but does not affect the length of the program.

8. For each construction of Ada language, which is called generic instantiation, the length increases by one. This reduces the length compared with the human-oriented strategy. Impact on counting vocabulary depends on context. The difference may be of any sign, but it should be a small within the compilation unit.

Now we can move on to solve our second task

Consider a compilation unit. Introduce the following designations:

$n1$ – the program vocabulary, which is calculated using the compiler-oriented strategy,

$L1$ – the length of program, which is calculated using the compiler-oriented strategy,

$n2$ – the program vocabulary, which is calculated using the human-oriented strategy,

$L2$ – the length of program, which is calculated using the human-oriented strategy.

Statement 1. Almost always the case there is following inequality

$$n1 > n2 . \tag{3}$$

In order to make sure, it is enough to see above eight paragraphs.

Statement 2. Let the compilation unit contains no comments. It is probably the following approximate equality

$$L1 \approx L2 . \tag{4}$$

The proof is as follows. Above, we saw that with a change of one strategy to another the value of the length can change of any party by many factors. It seems that in most cases the relevant factors are random and independent. Then specified equality must take place in the sense of mean values.

Knowing what explains the difference between the two strategies, one can specify the conditions under which this difference will be minimal.

Statement 3. In order to inequality (3) reduced to the approximate equality, it is necessary to require the following.

1. Any body of the program unit must be realized as separately compiled unit.
2. All types declarations should be concentrated in declarations of some library packages, and others packages should be involves all subprograms and tasks.
3. In the same package declarations we should not meet arrays and enumerations. Aggregates should not used in a library package declaration.
4. If possible, the body of the library unit (or subunit) should not include simultaneously the expressions with parentheses, array declarations, aggregates and type conversions.

We will find little utility programs that meet all the conditions of this approval. Let us put another statement.

Statement 4. Consider those compilation units, in which number of declarations is limited by the same constant. If the vocabulary of the unit will be large enough, the value of error

$$\frac{|n2 - n1|}{n2} \tag{5}$$

will be small.

Proof of the last two allegations based on the viewing list of the differences between the strategies that we have given above.

Statement 5. Let the program does not contain comments and combinations of words from the lists of (1)-(2) (or provides them little). Then accuracy of equality (4) should be high. This assertion is evident. It can be enhanced by adding other conditions, which exclude the practical differences between the two strategies.

Testing

In general, prior statements should not be supplemented accurate estimates. But they can be supplemented by the consideration of examples.

We use the fact that the authors of the compiler-oriented strategy gave two examples of calculations in his work [7]. We conducted calculation of the human-oriented strategy with the help of our application, which is described in [5]. The results are contained in Table 1.

Table 1
Comparison of the results for the two strategies

Name in Appendix B	$n1$	$n2$	$L1$	$L2$
Example 1 (NRPCA1)	54	47	191	203
Example 2 (OPCEA1)	48	46	203	205

We do not have the instrument for the measurements according the strategy of [7]. Therefore, the analysis was limited to eight examples compilation units, which took many forms prescribed in the Ada language. The size of each unit ranged from 15 to 150 lines of code. Differences in the size of the dictionary accounted for 5-20%.The length varied within 7.5%.

Conclusion

Although considered counting strategies based on different principles, the results of their application strictly comparable. It is probably because, in both cases, these principles have been carefully aligned with the standard description of programming language syntax.

An interesting consequence is the following. The compiler-oriented strategy was designed so that the Halstead metrics allowed assessing the resources necessary to Ada compiler. But a measurements using human oriented strategy are well correlated with measurements of the compiler-oriented strategy on a broad class of compilation units. This suggests that the property of the compiler-oriented strategy can be seen as a test for counting strategies that can be offered to other modern programming languages.

References

1. IEEE Std982.2-1988. IEEE Guide of the Use of Standard Dictionary of Measures to Produce Reliable Software. – [Electronic Resource]. – Access mode: http://members.aol.com/geshome/IEEE982/IEEE9822.pdf.
2. Halstead M.H. Elements of Software Science. New-Work: Elsevier, 1977.
3. Halstead M.H. Elements of Software Science / Translation from English to Russian by V.M. Yufa. – M.: Finance and Statistics, 1981. – 128 p.
4. Mishchenko V. O. Mathematical model of style Software Science for the metric analysis of complex scientific programs // Bulletin of V. Karazin Kharkiv National University, 2004. – № 629. – P. 105-111.
5. Series “Mathematical Modeling. Informational Technologies. Automated Control Systems” Issue 3. – P. 70-85.
6. Mishchenko V. O. One experiment in using energy metrics proposed for software process assessment // Radio-electronic and computer systems. – 2007. – № 8. – P. 121-124.
7. Mishchenko V. O. Energy Analysis of Software with examples of how it is applied to Ada programs. – Kharkov: Karazin KhNU, 2007. – 9 p.
8. D.M. Miller, R.S. Maness, J.W. Howatt, W.H. Shaw A software science counting strategy for the full Ada language //ACM SIGPLAN Notices, May 1987. – V. 22, Is. 5, ISSN:0362-1340. – P. 32-41.

Надійшла до редакції 11.02.2008

Рецензент: д-р техн. наук, проф. Б.М. Конорев, Національний аерокосмічний університет ім. М.Є. Жуковського «ХАІ», Харків.