UDC 004.412.2:(004.94+004.438)

Energy Metrics: an experience in assessment of programs developed within a mathematical package

AV Borovinskiy[1], AV Gakhov[2], VO Mishchenko[1]
[1]*V. Karazin Kharkiv National University, Ukraine*
[2]*ferret go GmbH, Germany*

The Energy Analysis for software systems is a complex of static analysis methods aimed to assess software reliability, complexity and benefits. It is based on a range of consistent thermodynamic analogies and uses experience from successes and failures of M. Halstead metrics. An important feature of this direction is that it can be applied to various forms of programming calculations. However, every new form requires to re-define the primitive characteristics needed for evaluation of meaningful metrics, and such definitions are already known for some traditional programming languages. In this work we have investigated XML format of PTS Mathcad and defined such its primitives as length, vocabulary and potential volume

Key words: *software quality metrics, static methods, Energy Analysis, computer algebra systems, Mathcad, XML, energy balanced software.*

Енергетичний аналіз програм - комплекс методів статичного аналізу, націлених на оцінку надійності, складності та переваг програмних систем. Він створений на основі послідовних термодинамічних аналогій, виходячи з досвіду удач і невдач метрик М. Холстеда. Важливою рисою цього напрямку є те, що він може застосовуватись при різних формах програмування обчислень. Однак для будь-якої нової форми потрібно заново визначати примітивні характеристики, необхідні для оцінювання змістовних метрик. Такі визначення відомі для деяких традиційних мов програмування. У роботі досліджено XML формат пакету PTS Mathcad и визначено такі примітиви як довжина, словник і потенційний об‘єм.

Ключові слова: *метрики якості ПЗ, статичні методи, енергетичний аналіз, математичні пакети, Mathcad, XML, енергетично збалансована програма.*

Энергетический анализ программ – комплекс методов статического анализа, нацеленных на оценку надёжности, сложности и преимуществ программных систем. Он создан на основе последовательных термодинамических аналогий, исходя из опыта удач и неудач метрик М. Холстеда. Важной чертой этого направления является то, что он применим при разных формах программирования вычислений. Однако для всякой новой формы нужно заново определять примитивные характеристики, необходимые для оценивания содержательных метрик. Такие определения известны для некоторых традиционных языков программирования. В работе исследован XML формат пакета PTS Mathcad и определены примитивы длины, словаря и потенциального объёма.

Ключевые слова: *метрики качества ПО, статические методы, энергетический анализ, математические пакеты, Mathcad, XML, энергетически сбалансированная программа.*

1 Introduction

The Energy Analysis for software systems is a branch of software quality control [1] that consists of static analysis methods aimed to assess the reliability, complexity and benefits of such systems. These methods can be considered as the modern development of M. Halstead’s Software Science ideas [2]. The direction of such evolution has been inspired by the theoretical thermodynamics [3]. Along with this, the Energy Analysis is based on the experience gained from investigation and usage of

M. Halstead's metrics (e.g. [4-6]) as well as metrics of the Energy Analysis itself [7-11].

The important advantage of both the Halstead's metrics and the Energy Analysis one is their independency of the software representation, whichever it is - textual, hierarchical, or even graphical. The downside is that, for each programming system, there is a need to develop (or even strictly define) a set of primitive characteristics to base metrics on. It is not a trivial problem even for "classical" procedural programming languages, mainly because of certain level of solution uncertainty. So, for new programming languages, the definitions have to be coordinated with already developed ones. This task has been solved for early-generation programming languages (60's-70's of the last century) and for modern languages, such as Ada, C++, Fortran, and Java [2, 4, 12-14, 11, 10]. Recently, such methods have been developed for such modeling languages as UML [15], but they are not ready for practical applications at this stage. For the XML-based sources, the Energy Analysis metrics have not been applied yet, but such problem has already raised for Android applications [10]. In [16] and [17], Halstead's metrics were used to evaluate projects that contained XML sources, however, in the first work, its authors concluded that such metrics are inappropriate for their use case; and in the second work, the metrics were used for the non-XML sources only.

Some custom forms of programming could also be found in popular general-purpose computer algebra systems. In this article we consider one of such systems - PTS Mathcad 14 [18] that offers a WYSIWYG interface and the ability to utilize a total user experience in the form of a file that contains both the program and the results of calculations.

The main goal of the current research is to develop ways for the automatic evaluation of Mathcad programs in order to estimate costs of creation, risks of programming errors, understandability, and forecast the maturity level. Note, that the one-file form of the Mathcad programs simplifies formulas for the energy metrics and makes some of them close to Halstead's metrics.

As the metrics which help to achieve our goal, we've selected the following: "difficulty measure" and "volume" (both are introduced by M. Halstead), "development difficulty", "programming work" (in this case they are just a rectification of the Halstead's metrics "difficulty" and "effort"), "specification energy" (evaluated from the Halstead's "approximation effort", but in case of Mathcad isn't equal to it), and, finally, "energy balance", that is specific for the Energy Analysis.

To estimate such metrics, we use the generic estimation schemes from the Energy Analysis [1, 3, 4] together with specific methods for Mathcad programs to define primitive characteristics (or simply - *primitives*).

- The first from such primitives is the *observed program length* [5] or simply *length* N - number of used semantic atoms counting all their occurrences. Halstead called such atoms as *tokens*. The second - *program vocabulary* or simply *vocabulary* η - the alphabet of unique programming symbols. The third (but last in sense of its importance) primitive is the differentiation of the units and the vocabulary to the sum of *operators* and *operands*. Such differentiation was very native in Halstead's times, but nowadays it quite often hits contradictions that is reported by many researches [1,3,4]. Finally, the fourth (but the most important one) – *architecture* *temperature*, that can be seen as the Halstead's *potential volume* in such a few cases when it can be defined uniquely [1-4]. In most cases, we call the architecture temperature as a *potential volume* and define as *V** just to keep the continuity of the terminology.

- According to our goal, the main tasks of the research can be formulated as follows:
- to develop methods for estimation of the primitives for Mathcad programs;
- to develop a way for automatic calculation of the energy metrics for Mathcad programs;
- to check if the expected regularities for Halstead's and energy metrics are fulfill on samples of programs from different authors and sources.

2 Application domains and regularities related to the metrics

We start by refreshing the well-known definitions of the Halstead's metrics - *volume*, *difficulty* and *effort*:

$$V = N\log_2 \eta \quad \text{or} \quad B = V/V_0\,, \quad (V_0 = 3000\ bit \cdot sym), \tag{2.1}$$

$$\hat{D} = \frac{N_2 \cdot \eta_1}{\eta_2 \cdot \eta_1{}^*}, \quad (\eta_1{}^* \text{ - is a constant equals to 2 } bit{\cdot}sym) \tag{2.2}$$

$$\hat{A} = \hat{D} \cdot V\,, \tag{2.3}$$

where N, N_2 - number of tokens and operands in the program (or program module);

η, η_1, η_2 - vocabularies of tokens, operators, operands.

Note, that there is no standard selection for names and denotations of the Halstead's metrics. In our work we use names from IEEE [5], but denotations are taken from [2, 4]. In (2.1) the alternative dimensionless metric *number of errors B* is the only metric in the strictest sense, but at the same time the Halstead's volume is playing an important role in definitions for other metrics, therefore, we can assume that (2.1) defines 2 different forms of the same metric.

M. Halstead introduced the following elementary primitives:

$$N, N_1, N_2, \eta, \eta_1, \eta_2 \tag{2.4}$$

but there are only 4 functionally independent among them. However, how it was discovered by Halstead and confirmed by other researchers on many examples, such primitives have another, statistical, connection - there is a correlation between N and the value that the standard [5] calls *estimated program length*:

$$\hat{N} = \hat{N}(\eta_1, \eta_2) = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2\,. \tag{2.5}$$

In many experiments performed in 70's of the last century with samples of programs written in Algol, Fortran, PL/1, etc., it was observed a close statistical relationship between N and $\hat{N}$. For instance, on samples of 10-20 [2] the values of the correlation coefficient were around 0.95 and more. As reported by another researcher in [4], with the large sample (about $1.6{\cdot}10^3$) the relative difference between these values for average-sized modules was less than 6%. Therefore, in experiments one shouldn't use more than 3 different Halstead's metrics, based on the primitives (2.4), at the same time.

As a guide to clarify the primitives' definitions for new languages could be used the fulfilment of the Halstead's program length equation [15]. However, as it followed from the English prose example [2], in a general case the *Halstead's program length equation* should be considered in the following generic form:

$$\exists k\ \forall S : Corr\left(N, \hat{N}(\eta_1, \eta_2); S\right) > 0.5\,, \tag{2.6}$$

where S – a large sample of programs (modules) writing on a some language (at least 20 modules, according to [2]);

$Corr(x, y, S)$ - correlation coefficient between x, y on sample *S*;

(Note, that the universal quantifier $\forall$ is considered here not literally, but in the sense that for any random sample of different modules it's unlikely that this condition will be violated). At the same time, it's appropriate to estimate the Halstead's *redundancy factor k* [2] by minimizing the average error *U*(*k*) of the module's length prediction over its estimation for the base sample S:

$$U(k) = \underset{S}{Avr}\left\{\frac{\left|\hat{N}(k\eta_1, k\eta_2) - N\right|}{N}\right\} \xrightarrow[k]{} \min. \tag{2.7}$$

where Avr – the sample's average.

In complex software systems that consist of many modules the Halstead's volume has to be corrected according to the intermodule connections and order of their development. Such correction could be achieved by introducing metric *volume of development*. The *program difficulty* metric becomes problematic in this case because it was proposed [2] in assumption of the module's self-sufficiency, thus, in the Energy Analysis it's considered as an alternative value to avoid errors in module's difficulty analysis (the *effort*, that derive from this metric, is totally ignored by us). In such cases when programs on the certain programming language can't be multi-module, then the importance (2.2) for control purposes is increasing, but the metric (2.3) isn't considered anyway during the energy analysis, except technical needs.

Finally, the last primitive characteristic - the *potential volume:*

$$V^* = \eta^* \log_2(\eta^*)\quad , \tag{2.8}$$

or, up to evident mathematical transformation, η^* – the number of the block's formal operators, that in the Energy Analysis is defined as:

$$\eta^* = 2 + p_1 + p_2 + \sqrt{j_1 \cdot j_2}\,, \tag{2.9}$$

where p_1 – number of configuration parameters;

p_2 – number of calling parameters (incl. output of the function, if any);

j_1 – number of I/O operations;

j_2 – number of files used by I/O operations.

The correctness of such definition opens the way to use in practice of the Energy Analysis the formula of M. Halstead's "theoretical" metric of difficulty:

$$D = W/V^* \quad , \tag{2.10}$$

where W – *development volume* for the module (equal to V, if only 1 module exists).

The *programming effort* metric implements the Halstead's idea that the programmer's effort related on the difficulty of the program understanding multiplied by the potential number of programming errors it may contain:

$$A = D \cdot W \tag{2.11}$$

The *specification energy* for a single module with many non-grouped blocks (as it takes place in Mathcad) can be found as:

$$E = \begin{cases} (V^*)^3 \big/ \lambda^2 & when\ m < 5 \\ (V^*)^3 \big/ \lambda^2 (\frac{m}{9} + 0.5)^2 & when\ m \geq 5 \end{cases} \tag{2.12}$$

где m – the number of non-grouped blocks of the module;
λ – the language level by Halstead.

As soon as everything above is calculated, we can estimate the most important metric in the Energy Analysis, the *energy balance*:

$$q = \frac{E - A}{\max(E, A)} \quad , \tag{2.13}$$

This metric provides the prediction of the balance between architecture design process and the process of design implementation in the code during the software development. Particularly, in the decimal representation of the metric (2.13) the first significant digit should be different from 9 for the well-balanced case.

An important regularity can be seen on random samples of programs that are developed in the same language (sometimes with additional external restrictions, e.g. also developed for the same application domain) – values (2.13) are randomly scattered around zero, however, with significant deviations. Such regularity was noticed, in fact, by M. Halstead for single-module programs without internal blocks [2]. However, he had hypothesized the value $(V^*)^3 \big/ \lambda^2$ as an approximation to effort estimation that wasn't confirmed afterwards [4]. Similarly, to the program length equation in form of (2.6), such regularity, in general, requires a normalization constant. As such constant we can use λ (2.12), however, then its value may be different from the Halstead's language level [19], that is evaluated on the basis of other considerations.

3 Estimation of elementary primitives

The concept of *tokens* in the source codes has been introduced by M. Halstead [2] to define alternative choices available to the software developer on each step of the development. For classical programming languages, as candidates on such role could be considered keywords and operation signs (incl. inseverable combinations), numerical literals, simple identifiers, etc. However, there are additional nuances for software development using general-purpose computer algebra systems (or simply, mathematical packages). Such development process could be followed by the immediate interpretation of inserted commands, simultaneous results output, it could also use graphics (mostly, produced by the developer and don't consist of sequences of some limited alphabet, but also could contain complex mathematical formulas). A natural representation of such structure is a hierarchy. The developer of a computing program for some mathematical package usually consequently selects tokens to put into the program, and, in fact, decides at each step whether to continue putting them at the current hierarchical level, create a new (deeper) level, or to finish the current layer (and maybe some other existing layers as well). Hierarchical structures can be mapped into some sequential text, for instance, using such mapping formats as XML and JSON. A Mathcad program is stored in .xmcd file that contains the XML representation, included a metadata for its execution in the Mathcad environment and the logic of the programmed calculations. The choice between the alternatives during the running process of Mathcad calculations is hard to describe formally, but it can be compared with the choice which would have to make the developer while building the logical part of .xmcd file for the developed program manually instead of using the Mathcad environment. We *hypothesize* that these two development processes (real and imaginary) are equivalent in terms of quality characteristics reflected in energy metrics (assuming that imaginary process is performed by a person, who also is imaginary and is as good in .xmcd files arrangement, as the real developer is in Mathcad).

Based on this hypothesis, we have built a projection of the “full” language for .xmcd files on such its part that is responsible for calculations by user's algorithm, that also allows us to transform .xmcd files into a hierarchically structured text. According to the hypothesis, such the algorithm's representation is the formalized Mathcad program. The process of obtaining such representation we call the .xmcd *cleaning*. It includes deleting all tags and attributes that are additional to computation itself, such as XML metadata, references to colors, images positions, etc. The adapted file must consist only of constructions of the following types, which contain exactly one token:

<regions> *text* </regions> , (3.1)

<ml: keyword {ml-option} > text **</ml:** keyword>, (3.2)

<result > *text* </result > , (3.3)

<ml: *operator* > , (3.4)

where elements written in italics are not parts of the tokens and added only to explain the ltoken usage. It must be clear that *text* may consist of the same or “other” lexemes of the formal language of adapted.xmcd files. The mentioned “other” tokens include such token kinds as

identifies , integer numbers , real-number , strings , (3.5)

and also:

1) the keywords that enter the **ml**-tags by pairs, such as

define , id , real , function , boundVars , (3.6)

and dozens of other;

2) ml-options such as

xml:space="preserve" , symbol=*string* , *[prefix-]*algorithm=*string* , (3.7)

and others (specific to the **ml**-tag's keywords);
3) some prefixes and postfixes of some ml-options, e.g.

auto- (ethe prefix of the last option, which shown in (3.7)) ; (3.8)

4) ml-operators such as

plus/ , div/ , neg/ , pow/ , greaterThan/ , (3.9)

and many others.
For example, consider such .xmcd file:

```
<region region-id="5" left="12" top="50.25" width="55.5" height="27.75" align-x="47.25" align-y="66" show-border="false" show-highlight="false" is-protected="true" z-order="0" background-color="inherit" tag="">
    <math optimize="false" disable-calc="false">
        <ml:define xmlns:ml="http://schemas.mathsoft.com/math30">
            <ml:id xml:space="preserve">lambda</ml:id>
            <ml:apply>
                <ml:div/>
                <ml:real>1</ml:real>
                <ml:real>10</ml:real>
            </ml:apply>
        </ml:define>
    </math>
    <rendering item-idref="5"/>
</region>
```

The result of its cleaning will be a text file that contains:

```
<ml:define>
    <ml:id xml:space="preserve">lambda</ml:id>
    <ml:apply>
        <ml:div/>
        <ml:real>1</ml:real>
        <ml:real>10</ml:real>
    </ml:apply>
</ml:define>
```

The cleaning process is implemented in EA_XMCD_Analyzer, developed by us in PHP that has useful features for such tasks [20]. To calculate elementary primitives (2.4) we use SAX-parser [21] (Simple API for XML), from the libxml library included as an extension in PHP and enabled by default. For instance, to search opening and closing tags corresponding to programming symbols-operators, we use built-in function *xml_set_element_handler*, and to search symbols-operands we use *xml_set_character_data_handler*.

4 Estimation of potential volumes

The adapted code in a file of some Mathcad program is the only module of this program, but its internal architecture is usually nontrivial. Blocks could be represented by such constructions as functions, integrals, derivatives, etc. For instance, the construction below defines a block - the function, that returns a value and one explicit parameter:

```
<ml:function>
    <ml:id xml:space="preserve">Init_Array</ml:id>
    <ml:boundVars>
        <ml:id xml:space="preserve">sep</ml:id>
    </ml:boundVars>
</ml:function>
<ml:program>
    …..
    <ml:return>
        <ml:id xml:space="preserve">vec</ml:id>
    </ml:return>
</ml:program>
```

There are 2 formal parameters for this block: the first is defined by the tag <ml:boundVars>, and the second – by the output. We also use PHP to find such structures and calculate all their parameters.

It’s more complex question how to deal with files and input/output operators, because the Mathcad semantics differs from the traditional procedural programming languages. In early versions of EA_XMCD_Analyzer we didn’t consider such case at all.

First of all, in the adapted .xmcd files we can consider as input/output operators the structures of assignments and calls, that were used by the developer to set values for constants and variables in Mathcad. As an example, review the following XML:

```
<ml:define>
  <ml:id xml:space="preserve"> a0 </ml:id>
  <ml:real> 1.0 </ml:real>
</ml:define>
```

Such structures could be also found inside of functions:

```
<ml:localDefine>
  <ml:id xml:space="preserve"> max </ml:id>
      <ml:apply>
         <ml:absval/>
         <ml:apply>
           <ml:indexer/>
           <ml:id xml:space="preserve"> v </ml:id>
           <ml:id xml:space="preserve"> c </ml:id>
          </ml:apply>
      </ml:apply>
</ml:localDefine>
```

The lexeme/paired tags <result> text </result> is to be considered as the output statement because they are generated in Mathcad source code when the program developer writes "<variable> =", having in mind to output the current value of a variable.

In the second version of the program EA_XMCD_Analyzer, the input/output operations were covered by the identification of constructions that are similar to described above. That made possible to calculate the value j_1 required in the formula (2.9), while the value of j_2 is chosen of 2, 1 or 0 depending on the presence or absence of constructions for keyboard input or display output.

5 Calculation

In the preceding sections, we have described the developed methods; and further we explain how they constitute the technology of automated assessment of the energy metrics (Fig. 1).

At the beginning, the selected (uploaded) text file ***progname.xmcd*** (where *progname* is the analyzed file's name) is subjected to the purification procedure described in section 3. The resulting **adapted_xmcd-*progname*.xml** file with a Mathcad computing program is the source of data for two processes. The first process calculates the elementary primitives (2.4) and dumps them into a text file **tokens_xmcd-*progname*.txt**. The second - finds formal parameters of blocks and calculates the potential volume. Data contained in the two files allow (Fig. 5.1) to calculate all Halstead's (2.1)-(2.3) and energy (2.10)-(2.13) metrics discussed above.

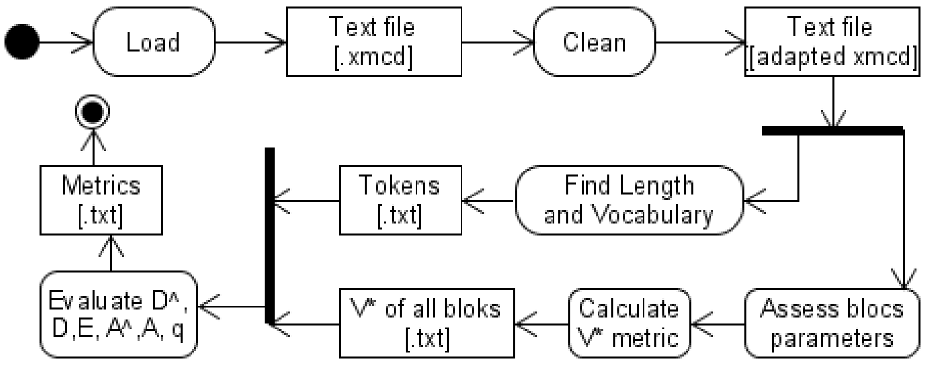

Fig. 5.1 The energy metrics assessment bases on .xmcd file

Out of these results we build the file **sss_xmcd-*progname*.txt**.

6 Check expected regularities for the metrics on an experimental sample

Despite the fact that for the Energy Analysis purposes splitting of specific program lexemes into statements and operands does not matter, it is interesting to compare the lexical features of different programs using the Halstead's length equation. We have selected those programs, which implement different computational methods. Table. 1 illustrates the results of primitives assessment (2.4) for the sample *S*23 of 23 Mathcad programs, selected from sources [18-21]. The table shows characteristics obtained for programs with the maximum and minimum length, as well as the average for the sample (an asterisk marks the average values defined as medians; in all other cases they are the arithmetic average).

We have been noticed the systematic and meaningful violation of the length equation (in its original form) for almost every considered Mathcad program (see the 2nd column in the Table 1), but the requirement (2.6) has been met.

Table.1 Reference values for the Mathcad programs in sample S23

The case	***N/N^***	***N*$_k$*^/N***	***B***	***D^***	***λ^***	***λ****	***λ***	***q*$_λ$**
max values	3.00	3.69	38.2	230	300.1	1073	4.36	0.96
avr values	0.68	0.85	7.5	69	2.19*	128,2*	0,53*	-0.98
min values	0.19	0.24	0.7	10	0.49	0,18	0.12	-0.26

The length N_k^ (the 3rd column in the Table 1) is obtained for the redundancy factor that was found from the condition (2.7):

$$Corr(N,\hat{N};S23)=0.51, \quad k=1.2 \quad (U(k)=0.63) \quad . \tag{6.1}$$

Thus, in this case we deal not with the redundancy, but with the lack of the alphabet; and the average N by N_k^ "prediction" error is 63%, which is worse than the 40% of the worst cases for the procedural languages [4] . We want to note that if to swap the roles of N and N_k^ in (2.7), the best value will be different: k = 2.12 (average error equals 60%).

It is also valuable to find out how big is the difference between assessment of difficulty classes in terms of metric (2.2) and the reference values calculated long time ago (1979) for the completely different language (PL/S). For the sample $S23$ we've obtain:

$$Avr(N_2/n_2)+Std(N_2/n_2)=7.52, \quad Avr(n_1)=27.87, \quad Std(n_1)=9.90, \tag{6.2}$$

where $Avr(x)$ is the average value of the sample x;

$Std(x)$ - is the standard error of the same sample.

Then, following the known procedure, detailed in [4], it's possible to calculate the boundaries separating the difficulty classes, both standard and optional [9] (for comparison, in brackets are given the known estimations for such boundaries calculated for the language PL/S [4, 5, 9]):

$$D_1=105\ (115), \quad D_2=142\ (160), \quad D_3=216\ (250),$$

$$D_4=291\ (340), \quad D_5=365\ (430)\ . \tag{6.3}$$

We can see that there is no fundamental difference between the obtained assessments for Mathcad programs and the corresponding known values. Also, 91% of the programs in the sample $S23$ belong to the difficulty class 0, only one - to the class 1, and the most difficult program belongs to the class 3. This is in contrast to the known examples of program modules written in universal procedural languages (e.g. [9]). It could happen that we chose mostly simple programs, but also it is possible that Mathcad is not intended for complex calculations more typical for universal procedural languages. If the result is not just a coincidence, we can expect that the majority of the programs in the sample, being simple, were well-thought-out by their authors and, hence, they are energetically balanced.

Let us try to find out the energy balance metrics distribution over our experimental sample. The values of language level λ lay in the range from 1.0 to 2.0 for high-level languages and for low-level ones between 0.5 and 1.0 [2]. For the Mathcad language we would expect $\lambda \approx 1.0$, but its estimation obtained over the sample $S23$ by the standard assessment method ($\hat{\lambda} \approx 2.2$) [4] and, especially, by the Halstead's "theoretical" method ($\lambda^* \approx 128$) are suspiciously big (Table 1). Assuming that $S23$ is representative with respect to the language level and reflects a regularity for the metric q (2.13) mentioned at the end of Section 2, it makes sense to consider for all programs such expression

$$\lambda = \sqrt{E(1)/A} \ , \qquad (6.4)$$

where $E(\lambda)$ is an estimation of (2.12), in which language level λ is indicated in parentheses (e.g. 1 in (6.4)). If to take for the language level the median of these values, then exactly the half of the programs will have $q > 0$, and for another half $q < 0$, which is unlikely. We performed a different (subjectively selected) procedure: to exclude 20% of the biggest and smallest values obtained by (6.4). The rest gave a fairly reliable first interval approximation to the expected language level:

$$0.314 < \lambda < 1.385 \quad . \qquad (6.5)$$

Unfortunately, this range is too wide. Let us consider its center $\lambda_1 = 0.849$; then – the widest range of λ variations, in which the deviation from λ_1 in both directions (bigger or smaller) doesn't change the interrelation between the signs of metrics q in our sample. This is our "satisfactory" but rather hypothetical interval approximation:

$$0.845 < \lambda < 0.93 \ . \qquad (6.6)$$

(Note, when it's necessary to use a particular value, it makes sense to take the center - 0.89).

Such choice of the language level has reduced the number of energetically unbalanced programs to just 2 out of 23, that is consistent with the assumption of maturity of most programs in the sample. Also, 39% of the programs has $q_\lambda > 0$ that shows the effect of metrics values scatter around zero. If we choose for the level any value of the interval (6.5), the most of programs stay balanced. Let us note that if assume $\lambda = \hat{\lambda}$ then only two programs (9%) will have $q > 0$, and the half of programs will be marked as unbalanced (12). This hardly meets our expectations, but the reliable conclusion about the language level requires much more statistical data to be processed.

7 Conclusion

In this article we have shown that for programs developed in Mathcad it is possible to create an accurate definition of Halstead's primitives that follows common practice of Halstead's definitions and the definition of the potential volume from the Energy Analysis. The research has been done under the hypothesis about correspondence between XML-oriented representation of the program and its interactive development process in Mathcad.

Furthermore, we have demonstrated the possibility to estimate the primitives in an automatic mode and, consequently, to create a program solution for the developed assessment method.

The test of the method on an experimental sample for Mathcad programs gave preliminary conclusions about relatively weak manifestation of the regularity known as Halstead's length equation in such case. It makes sense to consider such regularity in the form (2.7) including redundancy factor that has been estimated by us at the level of 1.2 (or 2.12 in case of another choice criterion). Note, the value bigger than 1.0 indicates the "insufficiency" of the alphabet in the Halstead's sense [2].

The tendency of Mathcad programs to be "in average" energy balanced is detectable for any language level chosen according to our "semi-heuristic" estimation from the range (6.5).

In the article we have presented a new halstead-like estimation method for primitive characteristics of the programming process in a mathematical package environment.

For the first time, the Energy Analysis' methods have been implemented for a formal language, which is based on XML and reflects the calculation logic of the Mathcad package. In addition, these methods are developed in PHP and they allow, in principle, to provide estimation online services.

REFERENCES

1. Мищенко В. О. Математическая модель стиля Software Science для метрического анализа сложных наукоёмких программ / В. О. Мищенко // Вісник Харківського національного університету: Зб. наук. праць. – Х., 2004. – № 629. – С. 70–85. – Title in English : Mishchenko, V. O. (2004) The Mathematical Model in Software Science Style for Measurement of Complex Scientific Software. – Bulletin of V. Karazin Kharkiv National University, Series «Mathematical Modelling. Information Technology. Automated Control Systems», 629, 3.
2. Halstead, M.H. Elements of Software Science / Halstead, Maurice H. Elsevier Publications, N-Holland, 1977. // Operating and programming systems series. – NY : Elsevier Science Inc. New York. – ISBN:0444002057.
3. Мищенко В. О. CASE–оценка критических программних систем. Том 3. Оценка качества / В. О. Мищенко, О. В. Поморова, Т. А. Говорущенко ; под ред. Харченко В. С. – Х : Нац. аэрокосмический ун–т «Харьк. авиац. ин–т», 2012. – 201 с. . – Title in English : Mishchenko, V. O., Pomarova, OV, Govorushchenko, TA (2012) Case-assessment of critical software systems. Volume 1. Quality, ed. Charchenko, VS, Kharkiv: Natonal Aerospase University named after N.E.Zhukovsky "KhAI".
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. 982.2-1988 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. - Institute of Electrical and Electronics Engineers, 1989.

6. Al Qutaish. An Analysis of the Design and Definitions of Halstead's Metrics / Proceedings of the 15th International Workshop on Software Measurement (IWSM'05), Montréal, Québec, Canada, 2005, pp. 337-352.
7. В. М. Годунко В. М. Качество транслятора шаблонов динамических html страниц для Ada WEB серверов / В. М. Годунко, В. О. Мищенко, М. М. Резник, Д. В. Штефан // Радіоелектронні і комп'ютерні системи. – 2012. – № 5. – С. 225-229. . – Title in English : V.M. Godunko, V.O. Mishchenko, M.M. Reznik, D.V. Shtefan. Dynamic generation html pages for ada web server Radioelectronic And Computer Systems, 2012, 5.
8. Міщенко В. О. Моделі та характеристики обчислювального кластера, які допомагають визначати напрямки його подальшого розвитку / В. О. Міщенко // Вісник Харк. нац. ун-ту., – 2013. – № 1058. Сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління», вип. 21. – С. 122-131. – Title in English : Mishchenko, V. O. (2013) Bulletin of V. Karazin Kharkiv National University, – 2013. – Series «Mathematical Modelling. Information Technology. Automated Control Systems», 1058, 21.
9. Мищенко В. О. Метрики трудности в оценке надёжности инструментальных библитек и фреймворков / В. О. Мищенко Вісник Харк. нац. ун-ту., – 2014. – № 1133. Сер. «Математичне моделювання. Інформаційні технології. Автоматизовані системи управління», вип. 25. – С. 126-147. . – Title in English : Mishchenko, V. O. (2014) Difficulty metrics in assessing the reliability of tool libraries and frameworks. Bulletin of V. Karazin Kharkiv National University, Series «Mathematical Modelling. Information Technology. Automated Control Systems», Issue 1131, 25.
10. Мищенко В. О. Использование энергетических метрик при анализе качества Android приложений / В. О. Мищенко, А. Ю. Уваренко // Труды XVII Международного симпозиума «Методы дискретных особенностей в задачах математической физики» (DSMMPh-2015), 2015. – Харьков-Сумы: изд. Харківського національного університету імені В. Н. Каразіна, 2015, С. 173-176. . – Title in English : Mishchenko, V. O., Uvarenko, A Yu. (2015) Proceedings of the XVII International Symposium "Discrete singularities methods in mathematical physics", Kharkov-Sumy.
11. Мищенко В. О. Преимущества, затраты и риски модификации реализаций методов дискретных особенностей с целью оптимизации / В. О. Мищенко, В. Паточкин // Вісник Харківського національного університету імені В. Н. Каразіна серія Математичне моделювання. Інформаційні технології. Автоматизовані системи управління, 2015. – вып. 28. – С. 69–76. . – Title in English : Mishchenko, V. O., Patochkin, B. V. (2015) Optimization of the methods of discrete singularities: the benefits, costs and risks of implementation modifications. Bulletin of V. Karazin Kharkiv National University, Series «Mathematical Modelling. Information Technology. Automated Control Systems», 2015, 28.
12. nag_metrics - NAGWare Fortran Tools - f77 Tools [Электронный ресурс]. – Режим доступа: https://www.lrz.de/services/software/programmierung/toolpack/nag_metrics.html.

13. Miller D. M. A software science counting strategy for the full Ada language / D. M. Miller, R. S. Maness, J. W. Howatt, W. H. Shaw // ACM SIGPLAN Notices. 1987. – Vol. 22, № 5. – P. 32–43.
14. Mishchenko V. O. Does The Different Definitions Of Ada Program Tokens Have Significant Difference? / V. O. Mishchenko // Radioelectronic And Computer Systems. – 2008. – № 7 (34) – С. 103–106.
15. Годунко В. М. Особенности энергетических метрик UML диаграмм / В. М Годунко, В. О. Мишенко, А. В. Пасека // Вестник Харк. нац. ун–та., – 2013. – № 1058. Сер. "Математическое моделирование. Информационные технологии. Автоматизированные системы управления", вып. 21. – С. 13-19. . – Title in English : Godunko V. M., The features of the energy metrics of UML diagrams / Godynko V. M., Mishcenko V. O., Paseka A. V. (2013) Bulletin of V. Karazin Kharkiv National University, – 2013. – Series «Mathematical Modelling. Information Technology. Automated Control Systems», 1058, 21.
16. Lassila, Matti. Comparison of two XML query languages from the perspective of learners / Lassila, Matti; Junkkari, Marko; Kekalainen, Jaana // Journal of Information Science, 2015. – Vol 41. – N 5. – P. 584-59.
17. Ming-Chang Lee. Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance Comparison of two XML query languages from the perspective of learners / Ming-Chang Lee // British Journal of Applied Science & Technology, 2014. – N. 4(21). – P. 3069-3095.
18. PTC Mathcad [Электронный ресурс] : Режим доступа: http://www.ptc.com/engineering-math-software/mathcad.
19. Мищенко В. О. Компьютерное моделирование характеристик схем программных систем / В. О. Мищенко // Радиоэлектронные и компьютерные системы. – Харьков, 2010. – № 5 (46). – С. 158–164. – Title in English : Mishchenko, V. O. Computer Modeling Of Software System Schemes Characteristics. Radioelectronic And Computer Systems, 2010, 5.
20. PHP: Hipertext Preprocessor [Электронный ресурс] : Режим доступа: http://php.net. – Заголовок с экрана.
21. PHP: XML Parser functions [Электронный ресурс] / Mehdi Achour, Friedhelm Betz, Antony Dovgal, Nuno Lopes, Hannes Magnusson, Georg Richter, Damien Seguy, Jakub Vrana, et al. - ed. Peter Cowburn // PHP Documentation Group. - http://php.net/manual/en/ref.xml.php.
22. PTC Community: Electrical Engineering [Электронный ресурс] : Режим доступа : https://www.ptcusercommunity.com/community/mathcad/electrical_engineering/content.

Надійшла у першій редакції 29.03.2016, в останній - 25.04.2016.