Al-Qutaish, R. E. and Abran, A., "An Analysis of the Designs and the Definitions of the Halstead's Metrics", In Proceedings of the 15th International Workshop on Software Measurement (IWSM'2005), September 12-14, 2005, Montreal, Canada. pp. 337-352.

An Analysis of the Design and Definitions of Halstead's Metrics

Rafa E. AL QUTAISH
École de Technologie Supérieure,
University of Québec,
1100 Notre-Dame Ouest,
Montréal, Québec H3W 1T8, Canada
rafa.al-qutaish.1@ens.etsmtl.ca

Alain ABRAN
École de Technologie Supérieure,
University of Québec,
1100 Notre-Dame Ouest,
Montréal, Québec H3W 1T8, Canada
alain.abran@etsmtl.ca

Abstract

Some software measures are still not widely used in industry, despite the fact that they were defined many years ago, and some additional insights might be gained by revisiting them today with the benefit of recent lessons learned about how to analyze their design. In this paper, we analyze the design and definitions of Halstead's metrics, the set of which is commonly referred to as 'software science'. This analysis is based on a measurement analysis framework defined to structure, compare, analyze and provide an understanding of the various measurement approaches presented in the software engineering measurement literature.

Keywords: *Halstead's Metrics, Software Science, Software Measurement, Measurement Framework, Metrology.*

1. Introduction

A number of software measures widely used in the software industry are still not well understood [1]. Some of these measures were proposed over thirty years ago, and, like many measures proposed later, they were defined mostly in an intuitive and heuristic manner by their designers. Moreover, authors describe their proposed measures in their own terms and structure, since there is not yet a consensus on how to describe and document the design of a software measure. Of course, the lack of a common design approach has made it difficult for practitioners to assess these measures.

Abran *et al.* [1] recently revisited the McCabe cyclomatic complexity number, illustrating that there is still ambiguity in its design and interpretation. In their study, they used the software measurement analysis framework proposed in [2].

Halstead's metrics – or what are commonly referred to collectively as 'software science' [3] – are among the most widely quoted software measures. For example, researchers have used Halstead's metrics to evaluate student programs [4] and query languages [5], to measure software written for a real-time switching system [6], to measure functional programs [7], to incorporate software measurements into a compiler [8] and to measure open source software [9].

In this paper, we investigate the various elements of the design and definitions of Halstead's metrics based on the software measurement analysis framework [2].

The rest of this paper is structured as follows: section 2 presents a brief overview of the analysis framework used to analyze Halstead's metrics. Section 3 presents an overview of Halstead's metrics. In section 4, the design and definitions of Halstead's metrics are investigated using the analysis framework introduced in section 2. Section 5 contains a discussion on this analysis and a summary of our observations.

2. Analysis Framework: an Overview

Definitions of the terms that will be used in this paper are provided first; these definitions have been adopted from ISO 15939 [10] and the *international vocabulary of basic and general terms in metrology* (VIM) [11]:

Entity: Object that is to be characterized by measuring its attributes [10].

Attribute: Property or characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means [10].

Measurement method: Logical sequence of operations, described generically, used in quantifying an attribute with respect to a specified scale [10].

Measurement procedure: Set of operations, described specifically, used in the performance of a

particular measurement according to a given method [10].

Base measure: Measure defined in terms of an attribute and the method for quantifying it [10].

Derived measure: Measure defined as a function of two or more values of base measures [10].

Unit of measurement: Scalar quantity, defined and adopted by convention, with which other quantities of the same kind are compared in order to express their magnitude [11].

Scale: Ordered set of values, continuous or discrete, or a set of categories to which the attribute is mapped [10].

Scale type: Depends on the nature of the relationship between values on the scale. *Nominal*, *ordinal*, *interval* and *ratio* are the four types of scale defined and identified in ISO 15939 [10].

The analysis framework of measurement proposed in [2] is based on work by Jacquet and Abran in [12]. This analysis framework consists of four phases of the software measurement life cycle: *defining the context*, *designing the measurement*, *applying the measurement method* and *exploring the measurement results* [2], as in Fig. 1. This measurement framework can be used to investigate and verify existing software measures.

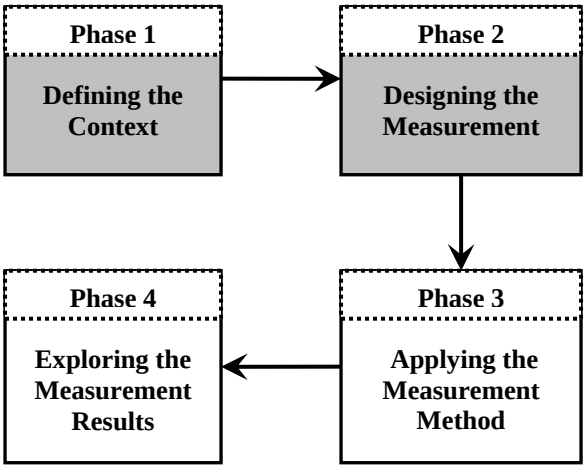

Fig. 1: The four phases of the Analysis Framework of Measurement proposed in [2].

To analyze the design and definitions of Halstead's metrics, we need to apply the first two phases of this analysis framework.

The two phases that will be used in this paper are summarized here. The first phase is *defining the context* in order to state the goals of the measurement that need to be investigated in more detail. In this phase, we have to select the objectives of the measurement in terms of the characteristics to be measured for a specific entity type [2].

The second phase, *designing the measurement*, is studied from three different points of view: activities, product and verification criteria. From the verification criteria viewpoint, this phase consists of three sub-phases [2]:

1- *The empirical and numerical worlds and their mapping*:

In order to define the empirical world, we need to determine the entities and their attributes to be measured. We should ensure that these attributes have been defined clearly and accurately, so that they are unambiguously characterized [2]. Then – for the numerical world defined – the selected mathematical structure should conserve the properties of that empirical world. This means that the mapping between the mathematical structure and the empirical world must produce the same form [2].

2- *The measurement method*:

Confirming and validating the numerical assignment rules (formulas) involve different activities, depending on the way those rules are expressed [2]. These formulas will be used to produce measurement values for the attributes to be measured. In addition, we have to validate the scale types of the measures and the units of measurement produced from the formulas based on the units of their operands.

3- *The measurement procedure*:

Verification of the measurement procedure to ensure that it constitutes a correct implementation of the measurement method. This verification should be achieved in accordance with the goals set out in the *defining the context* phase [2].

3. Halstead's Metrics: an Overview

According to Halstead, a computer program is an implementation of an algorithm considered to be a collection of tokens that can be classified as either operators or operands. In other words, a program can be thought of as a sequence of operators and their associated operands. All Halstead's metrics are functions of the counts of these tokens [13]. By counting the tokens and determining which are operators and which are operands based on a counting strategy, the following base measures can be collected [3]:

n_1: Number of distinct operators.
n_2: Number of distinct operands.
N_1: Total number of occurrences of operators.
N_2: Total number of occurrences of operands.

In addition to the above, Halstead defines [3]:
n_1^*: Number of potential operators.

n_2^*: Number of potential operands.

Halstead refers to n_1^* and n_2^* as the minimum possible number of operators and operands for a module or a program respectively. This minimum number would occur in a programming language itself, in which the required operation already existed (for example, in C language, any program must contain at least the definition of the function *main()*), possibly as a function or as a procedure; in such a case, $n_1^*=2$, since at least two operators must appear for any function or procedure: one for the name of the function and one to serve as an assignment or grouping symbol. Next, n_2^* represents the number of parameters, without repetition, which would need to be passed on to the function or the procedure [14].

All of the Halstead's so called "Software Science" metrics are defined based on the above collective measures (n_1, n_2, N_1, N_2, n_1^* and n_2^*).

Halstead defines the following metrics [3]:

- The *length* (N) of a program P is:

$$N = N_1 + N_2. \quad (1)$$

- The *vocabulary* (n) of a program P is:

$$n = n_1 + n_2. \quad (2)$$

- Program *volume* (V) is defined by Halstead in his book as:
  a) a suitable metric for the size of any implementation of any algorithm;[1]
  b) a count of the number of mental comparisons required to generate a program.[2]

  V can be computed using the following equation:

$$V = N * \log_2 n. \quad (3)$$

  The *length*, the *vocabulary* and *volume* of a program are considered as reflecting different views of program size [15].

- Program *potential (minimal) volume* (V^*), which is the volume of the minimal size implementation of a program P, is defined as[3]:

$$V^* = (2 + n_2^*) \log_2 (2 + n_2^*). \quad (4)$$

- Program *level* (L) of a program P with volume V is:

$$L = \frac{V^*}{V}. \quad (5)$$

  The program *level* emphasizes that growth in volume leads to a lower level of program, and conversely. The largest value for L is 1. In addition, this value is interpreted as referring to the most ideally written program and as measuring how well written a program is. Thus, programs with L values close to 1 are considered to be well written, in general L<1 [5].

- Program *difficulty* (D) is defined as the inverse of program level L:

$$D = \frac{1}{L}. \quad (6)$$

- The program *level estimator* ($\hat{L}$) of L is defined by Halstead as:

$$\hat{L} = \frac{2}{n_1} * \frac{n_2}{N_2}. \quad (7)$$

  and interpreted by Menzies *et al.* [14] and by Fenton and Pfleeger [16] as:

$$\hat{L} = \frac{1}{D} = \frac{2}{n_1} * \frac{n_2}{N_2}. \quad (7.1)$$

- The *intelligent content* (I) of a program P is a measure of the information content of program P, and is defined as:

$$I = \hat{L} * V. \quad (8)$$

- *Programming effort* (E) is a measure of the mental activity required to reduce a preconceived algorithm to a program P. E is defined as the total number of elementary mental discriminations required to generate a program:

$$E = \frac{V}{L} = \frac{n_1 N_2 N \log_2 n}{2 n_2}. \quad (9)$$

  In the effort definition, the unit of measurement of E is claimed by Halstead to be an *elementary mental discrimination*.

- The required *programming time* (T) for a program P of effort E is defined as:

$$T = \frac{E}{S} = \frac{n_1 N_2 N \log_2 n}{2 n_2 S}. \quad (10)$$

[1] Halstead's book [3], p. 19.
[2] Halstead's book [3], p. 47.
[3] No objective evidence documented in [3] that this is indeed a minimal implementation.

where S is the Stroud number[1], defined as the number of elementary discriminations performed by the human brain per second. The S value for software scientists is set to 18 [17]. The unit of measurement of T is the second.

All the above ten equations are based on the results of n_1, n_2, N_1, N_2, n_1^* and n_2^*, which themselves are based on a counting strategy to classify the program tokens as operators or operands.

Unfortunately, there is a problem in distinguishing between operators and operands. This problem occurs because Halstead has provided an example[2] with specific illustrations of operators and operands, but without generic definitions applicable to any program context. That is, Halstead has not explicitly described the generic measurable concepts of operators and operands. He has asserted only that – in the example he provides – their description is intuitively obvious and requires no further explanation. In practice, for measurement purposes, intuition is insufficient to obtain accurate, repeatable and reproducible measurement results.

Therefore, it is important that the counting strategy be clearly defined and consistent, since all Halstead's software science depends on counts of operators and operands [18]. However, there is no general agreement among researchers on the most meaningful way to classify and count these tokens [19]. Hence, individual researchers (and practitioners as well) must state their own interpretation or, alternatively, use one of the available counting strategies proposed by other researchers, such as in [20-23]. Furthermore, Li *et al.* have proposed rules for identifying operators and operands in the object-oriented programming (OOP) languages [24].

Of course, it is to be expected that different counting strategies will produce different values of n_1, n_2, N_1 and N_2, and, consequently, different values for the above ten equations.

4. Analysis of the Design and Definitions of Halstead's Metrics

4.1. Defining the Context

The objective of Halstead's metrics is to measure the following characteristics of a program: *length*, *vocabulary*, *volume*, *level*, *difficulty* and *intelligence content*. In addition, they are used to measuring what is referred to as "other characteristics" of the developer: *programming effort* and *required programming time*. All these measures are based only on the number of operators and the number of operands the given program or algorithm contains.

The last two attributes, which refer to a developer's attributes (*programming effort* and *required programming time*), seem to be identical, since 'effort to write a program' is similar to 'required programming time'.

4.2. Designing the measurement

4.2.1. The empirical and numerical worlds and their mapping

The entities that can be used to apply Halstead's metrics are the source code itself or the algorithm of that source code. However, applying Halstead's metrics to these two entities will produce different values for the same base measures. For example, in Java language, the number of operators in the source code is different from the number of operators in the equivalent algorithm for that source code, since – as an example – in Java source code, each statement must be end with a semicolon (;), which is an operator.

Halstead's metrics are based on two attributes: the number of operators and the number of operands. As mentioned in section 3, there is no agreement on how to distinguish between operators and operands. Therefore, different counting strategies will produce different numbers of operators and operands for the same program or algorithm. The two attributes can be easily mapped to a mathematical structure by counting the number of operators and operands in the program source code or the equivalent algorithm.

Furthermore, Kiricenko and Ormandjieva [25] investigated the validation of the representation condition for Halstead's program length metric.

4.2.2. The measurement method

To obtain a value for each of Halstead's metrics, ten equations have to be computed (see section 3). It is to be noted that all of these equations (equations 1 to 10) correspond to a 'derived measure', as defined by the *international vocabulary of basic and general terms in metrology* (VIM) and the ISO 15939.

Equation (3) is of a ratio scale type, while equation (5) is of an ordinal scale type, as noted by Fenton and Pfleeger [16]. By contrast, Zuse [26] maintains that equation (1) is of the ratio scale type

[1] In 1967, a psychologist, John M. Stroud, suggested that the human mind is capable of making a limited number of mental discrimination per second (Stroud Number), in the range of 5 to 20.

[2] Halstead's book [3], pp. 6-8.

and equations (2), (3), (6) and (9) are of an ordinal scale type. Moreover, it can be observed that equation (4) is also of the ratio scale type. However, it is not clear to which scale type equations (7), (8) and (10) belong.

These conclusions on the scale types of Halstead's metrics need to be revisited when the units of measurement in Halstead's equations are taken into consideration.

For instance, in equation (1), the program *length* (N) is calculated by the addition of the total number of occurrences of operators and the total number of occurrences of operands. However, since their units are different, operators and operands cannot be directly added together unless the concept common to them (and its related unit) is taken into consideration in the addition of these numbers, that is, 'occurrences of tokens': then, the right-hand side of equation (1) gives 'occurrences of tokens' as a measurement unit on the ratio scale:

$$N^{\text{occurrences of tokens}} = N_1^{\text{occurrences of operators}} + N_2^{\text{occurrences of operands}}.$$

From equation (2), the program *vocabulary* (n) can be constructed by adding the number of distinct operators and the number of distinct operands:

$$n^{\text{distinct tokens}} = n_1^{\text{distinct operators}} + n_2^{\text{distinct operands}}.$$

The measurement unit here is 'distinct tokens'. This measurement unit must then also be assigned to the left-hand side of this equation, labeled 'vocabulary', and associating it to the related concepts.

It can be noted that, while the concept of 'length' is associated with a number, the concept of 'vocabulary' is not. Indeed, the program vocabulary (n) reflects a different view of program size [15], and it is a measure of 'the repertoire of elements that a programmer must deal with to implement the program' [27]. Most probably, an expression such as 'size of a vocabulary' would have been more appropriate.

From equation (3), program *volume* (V) has been interpreted with two different units of measurement; 'the number of bits required to code the program' [17] and 'the number of mental comparisons needed to write the program' [14] on the left-hand side of the equation:

$$V^{\text{bits or mental comparisons}} = N^{\text{occurrences of tokens}} * \log_2 n^{\text{distinct tokens}}.$$

Thus, there is no relationship between the measurement unit on the left-hand side and those on the right-hand side of this equation. Furthermore, on the right-hand side, the true meaning of the multiplication of 'occurrences of tokens' and 'distinct tokens' is not clear. Such a multiplication would normally produce a number without a measurement unit, see Fig. 2.

In general, in engineering applications we do not take the logarithm of a dimensioned number, only of dimensionless quantities. For instance, in calculating decibels, we take the logarithm of a ratio of two quantities. A ratio of quantities with the same dimensions is itself dimensionless. We can write

$$log(a/b) = log(a) - log(b)$$

making it appear that we are taking the *logs* of dimensioned quantities (*a)* and (*b)*, but the dimensions come out in the wash: by the time we have finished (subtracting one *log* from the other), we have effectively taken the log of a dimensionless quantity, **(***a/b***)**.

We can regard units as factors in an expression, for instance:

8 meters = *8 * [1 meter]*
800 cm = *800 * [1 cm]*
= *800 * 0.01 * [1 meter]*

In these terms, we have:

$(8m)*log_2(8m) = 8*[1m]*log_2(8*[1m])$
$= 8*[1m]*(log_2(8)+log_2[1m])$
$= (8*log_2(8)+8*log_2[1m])*[1m]$

That inconvenient $8*log_2[1m]$ is an additive term that depends on the units being used. If it is part of a valid engineering calculation, this term will be canceled out somewhere in the process. It may be, for instance, that when we take the *log* of *8 meters*, we are actually taking the *log* of a ratio of *8 meters* to a *one-meter* standard length.

Fig. 2: Explanation of the measurement unit produced by $\log_2$[1].

Equation (4) gives the definition of the program *potential volume* (V^*), which is a prediction of the program volume:

[1] Contact by e-mail with Mr. Richard Peterson, The Math Forum (Ask Dr. Math) at Drexel University, http://www.mathforum.org/dr.math/.

$$V^* = (2^{\text{potential operators}} + n_2^{*\ \text{potential operands}}) \log_2 (2^{\text{potential operators}} + n_2^{*\ \text{potential operands}}).$$

In this equation, the value '2' was assigned to n_1^*, as seen in section 3. The measurement unit of the left-hand side is the same as in the previous equation (equation (3)), while there is no recognizable measurement unit for the right-hand side. As in equation (3), such a multiplication would also normally produce a number without a measurement unit, see Fig. 2.

The program *level* (L) can be calculated using equation (5), in which there is no measurement unit for the left hand-side, either from Halstead himself or from other researchers. In the sense that this is the correct structure for a ratio with the same unit in both numerator and denominator; the end result is therefore a percentage:

$$L = \frac{V^{*\ \text{bits}}}{V^{\ \text{bits}}} = \frac{V^{*\ \text{mental comparisons}}}{V^{\ \text{mental comparisons}}}.$$

For equation (6), the *difficulty* (D) is a measure of 'ease of reading' and can be seen as a measure of 'ease of writing' as well [27]. The right-hand side is also a percentage. What the right-hand side of equation (6) means is a riddle, as its associated label on the left-hand side.

In Equation (7), for the program *level estimator* ($\hat{L}$), there is no measurement unit for the left-hand side, while the right-hand side consists of a combination of four distinct measurement units. The exact meaning is again a riddle:

$$\hat{L} = \frac{2^{\text{potential operators}}}{n_1^{\text{distinct operators}}} * \frac{n_2^{\text{distinct operands}}}{N_2^{\text{occurrences of operands}}}.$$

In equation (8), referred to as the *intelligent content* of the program (I), there is no measurement unit on the left-hand side. For the right-hand side of this equation, the measurement unit of $\hat{L}$ – which is not known since it is a combination of units – is multiplied by the measurement unit of V:

$$I = \hat{L} * V^{\text{bits}} = \hat{L} * V^{\text{mental comparisons}}.$$

As for equations (6) and (7), the exact meaning of the left-hand side of equation (8) is a riddle if we attempt to interpret this number with measurement units.

Equation (9) is used by Halstead to compute the *effort* (E) required to generate a program:

$$E^{\text{elementary mental discriminations}} = \frac{n_1^{\text{distinct operators}} N_2^{\text{occurrences of operands}} N^{\text{occurrences of tokens}} \log_2 n^{\text{distinct tokens}}}{2^{\text{potential operators}} n_2^{\text{distinct operands}}}.$$

The measurement unit of the left-hand side of this equation, referred to as 'effort', would be expected to be something such as 'hours' or 'days'. Halstead, however, referred to 'the number of elementary mental discriminations' as the unit of measurement for the left-hand side. Next, in the sense that the 'distinct operators', the 'distinct operands' and the 'occurrences of operands' are, in a generic sense, 'tokens', then it can be concluded that the measurement unit of the right hand-side of this equation is a combination of measurement units. Therefore, there is no relationship between the units of measurement of the left-hand and the right-hand sides in equation (9).

Finally, equation (10) is used to compute the required *programming time* (T) for the program:

$$T^{\text{seconds}} = \frac{n_1^{\text{distinct operators}} N_2^{\text{occurrences of operands}} N^{\text{occurrences of tokens}} \log_2 n^{\text{distinct tokens}}}{2^{\text{potential operators}} 18^{\text{psychological moments per second}} n_2^{\text{distinct operands}}}.$$

Again, the measurement unit of the left-hand side, that is, seconds, does not in any way imply the measurement unit of the right-hand side, that is, a combination of many different measurement units. In view of the fact that, Halstead refers to the 'moments' in this equations as "the time required by the human brain to perform the most elementary discrimination"[1].

5. Discussion and Conclusions

In this paper, we have investigated a well-known set of measures – Halstead's metrics – by focusing on their design and, in particular, on their measurement units. The following comments can be made about Halstead's metrics:

- Based on ISO 15939 [10] and the international vocabulary of basic and general terms in metrology (VIM) [11], Halstead's metrics can be classified as six based measures (n_1, n_2, N_1, N_2, n_1^* and n_2^*) and ten derived measures (equations (1) to (10)).
- Halstead has not explicitly provided a clear and complete counting strategy to distinguish between the operators and the operands in a given program or algorithm. This has led researchers to come up

[1] Halstead's book [3], p. 48.

with different counting strategies and, correspondingly, with different measurement results for the same measures and for the same program or algorithm.

- There are problems with the units of measurement for both the left-hand and the right-hand sides of most of Halstead's equations.
- The implementation of the measurement functions of Halstead's metrics has been interpreted in different ways than the goals specified by Halstead in their designs. For example, the program length (N) has been interpreted as a measure of program complexity, which is a different characteristic of a program [15].
- Equations (6) and (7.1), using basic mathematical concepts, lead to $\hat{L}$ being identical to L; this point can be clarified as follows:

$$\hat{L} = \frac{1}{D} \quad (6), \qquad D = \frac{1}{L} \quad (7.1),$$

$$\hat{L} = \frac{1}{\frac{1}{L}}, \qquad (11)$$

$$\hat{L} = L.$$

Therefore, using Fenton's description of $\hat{L}$ [1], the program level estimator is identical to the program level.

- Using the previous observation (that is, $L = \hat{L}$), and from equations (5) and (8), it can be concluded that $I = V^*$. The clarification of this point is as follows:

$$I = \hat{L} * V \quad (8), \qquad L = \frac{V^*}{V} \quad (5), \qquad L = \hat{L} \quad (11),$$

$$I = L * V,$$

$$I = \frac{V^*}{V} \times V,$$

$$I = V^* = \text{size unit} \rightarrow \text{Bits} \text{ or } \text{Mental comparisons}$$

Therefore, how we can use the same value to measure both 'intelligent content' (I) and 'program potential volume' (V^*), two different attributes of a program or algorithm? Also, how do we give different units of measurement to the same value?

- A number of addition issues can be raised such as the following:
  Equations (9) and (10), which give the programming effort (E) and the required programming time (T) in seconds, do not take into account technology evolution and characteristics: for instance, new programming languages (i.e. the 4th generation programming languages) need less time for programming since most of the programming effort is expended by means of drag-and-drop processes, as in Visual Basic.

In summary, the Halstead metrics, as designed almost thirty years ago, do not meet a key design criterion of measures in engineering and the physical sciences. Further research is still required to address the weaknesses identified in their designs. In a follow-up research to the findings of this paper, [28] has investigated these issues and explored them from the perspective of the extensive structure from measurement theory. In doing so, a number of assumptions were made and will require further investigation.

6. Acknowledgments

The authors would like to thank Dr. Olga Ormandjieva and Mrs. Victoria Kiricenko for their comments and suggestions, and Mr. Richard Peterson from the Math Forum for his comments on the measurement unit of the logarithm results.

The opinions expressed in this paper are solely those of the authors.

References

[1] Abran, A., Lopez, M., and Habra, N., "An Analysis of the McCabe Cyclomatic Complexity Number", in *Proceedings of the 14th International Workshop on Software Measurement (IWSM) IWSM-Metrikon*, 2004, Magdeburg, Germany: Springer-Verlag, pp. 391-405.

[2] Habra, N., Abran, A., Lopez, M., and Paulus, V., "Toward a Framework for Measurement Lifecycle", in *University of Namur, Technical Report TR37/04*, 2004.

[3] Halstead, M. H., *Elements of Software Science*, 1977, New York: Elsevier North-Holland.

[4] Leach, R. J., "Using Metrics to Evaluate Student Programs", *ACM SIGCSE Bulletin*, Vol. 27, No. 2, 1995, pp. 41-48.

[5] Chuan, C. H., Lin, L., Ping, L. L., and Lian, L. V., "Evaluation of Query Languages with Software Science Metrics", in *Proceedings of the IEEE Region 10's Ninth Annual International Conference on Frontiers of Computer Technology TENCON'94*, 1994, Singapore, pp. 516-520.

[1] Fenton and Pfleeger book [16], p. 251.

[6] Bailey, C. T. and Dingee, W. L., "A Software Study Using Halstead Metrics", in *Proceedings of the 1981 ACM Workshop / Symposium on Measurement and Evaluation of Software Quality*, 1981, Maryland, USA, pp. 189-197.

[7] Booth, S. P. and Jones, S. B., "Are Ours Really Smaller Than Theirs", in *Glasgow Workshop on Functional Programming*, 1996, Ullapool, Scotland, UK, pp. 1-7.

[8] Al Qutaish, R. E., *Incorporating Software Measurements into a Compiler*, MSc thesis, Department of Computer Science, 1998, Serdang: Putra University of Malaysia.

[9] Samoladas, I., Stamelos, I., Angelis, L., and Oikonomou, A., "Open Source Software Development Should Strive for Even Greater Code Maintainability", *Communication of ACM*, Vol. 47, No. 10, 2004, pp. 83-8.

[10] ISO/IEC, *ISO/IEC IS 15939: Software Engineering - Software Measurement Process*, 2002, Genève: International Organization for Standardization.

[11] ISO/IEC, *International Vocabulary of Basic and General Terms in Metrology (VIM)*, 1993, Genève: International Organization for Standardization.

[12] Jacquet, J. and Abran, A., "From Software Metrics to Software Measurement Methods: A Process Model", in *the 3rd IEEE International Software Engineering Standards Symposium and Forum ISESS'97*, 1997, Walnut Creek, California, USA, pp. 128-135.

[13] Henry, S. and Kafura, D., "Software Structure Metrics Based in Information Follow", *IEEE Transaction on Software Engineering*, Vol. 7, No. 5, 1981, pp. 510-518.

[14] Menzies, T., Stefano, J. S. D., Chapman, M., and McGil, K., "Metrics That Matter", in *the 27th Annual NASA Goddard Software Engineering Workshop*, 2002, Greenbelt, Maryland, USA, pp. 51-57.

[15] Fenton, N., "Software Measurement: A Necessary Scientific Basis", *IEEE Transaction on Software Engineering*, Vol. 20, No. 3, 1994, pp. 199-206.

[16] Fenton, N. E. and Pfleeger, S. L., *Software Metrics: A Rigorous and Practical Approach*. 2nd ed., 1997, Boston: PWS Publishing Company.

[17] Hamer, P. G. and Frewin, G. D., "M. H. Halstead's Software Science - A Critical Examination", in *the Proceedings of the 6th International Conference on Software Engineering*, 1982, Tokyo, Japan, pp. 197-206.

[18] Lister, A. M., "Software Science - The Emperor's New Clothes", *the Australian Computer Journal*, Vol. 14, No. 2, 1982, pp. 66-70.

[19] Shen, V. Y., Conte, S. D., and Dunsmore, H. E., "Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support", *IEEE Transaction on Software Engineering*, Vol. 9, No. 2, 1983, pp. 155-165.

[20] Salt, N. F., "Defining Software Science Counting Strategies", *ACM SIGPLAN Notices*, Vol. 17, No. 3, 1982, pp. 58-67.

[21] Szentes, S., *QUALIGRAPH User Guide*, 1986, Budapest: Research and Innovation Center.

[22] Abd Ghani, A. A. and Hunter, R., "An Attribute Grammar Approach to specifying Halstead's Metrics", *Malaysian Journal of Computer Science*, Vol. 9, No. 1, 1996, pp. 56-67.

[23] Conte, S. D., Dunsmore, H. E., and Shen, V. Y., *Software Engineering Metrics and Models*, 1986, Menlo Park, California: Benjamin Cummings.

[24] Li, D. Y., Kiricenko, V., and Ormandjieva, O., "Halstead's Software Science in Today's Object Oriented World", *Metrics News*, Vol. 9, No. 2, 2004, pp. 33-40.

[25] Kiricenko, V. and Ormandjieva, O., "Measurement of OOP Size Based on Halstead's Software Science", in *Proceedings of the 2nd Software Measurement European Forum*, 2005, Rome, Italy.

[26] Zuse, H., *A Framework of Software Measurement*, 1998, Berlin: Walter de Gruyter.

[27] Christensen, K., Fitsos, G. P., and Smith, C. P., "A perspective on software science", *IBM Systems Journal*, Vol. 20, No. 4, 1981.

[28] Zuse, H., "Resolving the Mysteries of the Halstead Measures", *to be published in METRICON*, Fall of 2005, Germany.