Syntactic Complexity Metrics and the Readability of Programs in a Functional Computer Language

Klaas G. van den Berg

University of Twente, Faculty of Computer Science, PO Box 217, 7500 AE Enschede, The Netherlands

Abstract: This article reports on the definition and the measurement of the software complexity metrics of Halstead (1977) and McCabe (1976) for programs written in the functional programming language Miranda. An automated measurement of these metrics is described. In a case study, the correlation is established between the complexity metrics and the expert assessment of the readability of programs in Miranda, and compared with those for programs in Pascal.

Keywords: Software metrics, psychological complexity, syntactic complexity, readability, imperative programming, functional programming.

Introduction

Computer programs are written in order to carry out the solution to a problem by a computer. During the construction of programs there is a continual need to read and understand the text of programs: for the further development of the program to meet new specifications, for the modification of programs to correct errors, or to consider programs or program parts for possible reuse. The programs can be written in various programming languages, which may differ in their expressive power. The expressive power of a language is reflected in the ability to write programs that are both succinct and understandable (Fleck, 1990). Understandability or readability will be defined as the extent to which the function of the program and its components are easily discerned by reading the program text (Boehm, Brown, Kaspar, Lipow & Merritt, 1978). We will consider the readability of programs written in the functional programming language Miranda (Turner, 1986) and the imperative programming language Pascal.

The basic difference between functional and imperative programming languages lies in the hiding of the computational model (Petre & Winder, 1990). The imperative model incorporates the Von Neuman machine characteristics in the notions of assignment, state and effect. A characteristic of this language class is the explicit flow of control, e.g., sequencing, selection and repetition. In assignments, the value of memory places denoted with variables is changed during program execution. This model of operation by change of state and by alteration of variable values is also named 'computation by effect'. The functional model is characterized by 'computation by value'. Functions return values and have no side-effects, and expressions represent values. There is no notion of updatable memory accessible by instruction. The program consists of a script with a number of mathematical-like definitions and an expression that must be evaluated. Functions can be passed as arguments to other functions and can be the result of a function application (higher-order functions). These programs possess the property of referential transparency, which means that in a fixed context the replacement of a subexpression by its val-

ue is completely independent of the surrounding expression. Therefore functional programming is more closely related to mathematical activities (Bird & Wadler, 1988).

There is a growing interest in functional programming languages, because of their expressive power and the possibility to reason about correctness of programs. There are claims on the readability of functional programs, for example: 'In many cases the functional programming style yields more elegant and comprehensible programs than the imperative programming style' (Springer & Friedman, 1990) and 'Functional programming leads to programs which are exceptionally clear and concise and to the prospect of greatly increased software reliability and development speed' (Bailey, 1990). In a case study on the productivity of programming in a functional programming environment, some of these claims have been confirmed (Sanders, 1989).

Moreover, the learning of programming in a functional programming style should have advantages over learning in an imperative style (Springer & Friedman, 1990; Bailes & Salzman, 1989). At the University of Twente, first year students in Computer Science receive a course in functional programming, which forms the base of the programming curriculum (Joosten & van den Berg, 1990). An important aspect of teaching programming is to give feedback to the novice programmers on the readability of their programs and intermediate products during the design process. This readability can be assessed by teachers. It is also possible to measure internal attributes of programs based on the syntax of the program text that correlate with the readability of programs. These values can be used in feedback to the students, which may be interactively during the program development. The measures of software attributes are usually called software metrics and are described in the following section. In a subsequent case study we will report on the correlation between readability and the software metrics for programs in Miranda, and compare this with the correlation for programs in Pascal.

Software metrics

Software metrics are used to assess the process of the construction of software, the products of this process, and the use of human and machine resources (Conte, Dunsmore & Shen, 1986; Fenton, 1991). We will focus our discussion on the use of human resources in the interaction with software. This attribute of software is referred to as the psychological complexity (Curtis, Sheppard, Milliman & Love,1979), and we restrict ourselves to the readability or comprehensibility of the program text. The computational complexity, i.e., the efficiency of the use of the machine resources (time and memory space), will not be considered here.

There are several software metrics for the static syntactic complexity of program text described in the literature. We will use the software metrics based on Halstead's Software Science (Halstead, 1977) and the cyclomatic complexity number of McCabe (1976). The study of Curtis (1981) shows empirical evidence that the complexity metrics of McCabe and Halstead relate to the psychological complexity, as expressed in the difficulty in understanding and modifying software. The relation between syntactic complexity and cognitive complexity has been investigated by Khalil & Clark (1989). Shen, Conte & Dunsmore (1983) give a critical review of Software Science. They show that the effort E, as defined by Halstead (see below), correlates with the understandability of programs. However, the psychological aspects of the Halstead metrics have been criticized (Coulter, 1983).

The software metrics of Halstead and McCabe have been applied mainly to programs written in imperative programming languages. We have derived these metrics for the functional pro-

gramming language Miranda. The definition of these metrics for this language will be given in the section below, followed by a description of an automated measurement of the metrics.

Halstead and McCabe metrics: Halstead (1977) proposed in his theory of Software Science that some useful measures for computer programs can be derived from four basic metrics: the count of unique operands and operators, and their total frequencies. A symbol in a program that specifies an action is considered an operator, while a symbol used to represent data is considered an operand. Among the derived metrics are the program volume, the level of implementation and the programming effort.

The basic metrics are defined as the number of unique operators η_1, the number of unique operands η_2, the total number of operators N_1, and the total number of operands N_2. The vocabulary of a program is defined as $\eta = \eta_1 + \eta_2$, and the length of a program as the total number of tokens $N = N_1 + N_2$. The volume (size) of a program is defined as $V = N \times \log_2 \eta$. The potential volume V* of a program represents the size of the program in its most succinct form. Halstead showed that $V^* = (2 + \eta_2^*) \times \log_2 (2 + \eta_2^*)$, where η_2^* is the number of different input/output parameters. The program level, or the level of implementation, L is the ratio of the potential volume V* and its actual volume V, i.e., $L = V^* / V$. The effort to generate a program is defined by the relation $E = V / L$. Other quantities have been defined and relations between these quantities can be obtained by algebraic manipulation.

McCabe (1976) developed a measure of software based on the decision structure of a program. The program is represented as a graph G with a unique entry and exit point. The edges represent branches caused by a decision, and the nodes represent a piece of code. The metric counts the number of linear independent paths through a program and is also called the cyclomatic complexity number. This metric is related to the difficulty of testing a program. The cyclomatic complexity number is $v(G) = e - n + 2$. The number of edges in the graph is e and the number of nodes is n. It can be shown that v(G) is equal to the number of decisions in a program + 1, provided that all decision nodes have outdegree 2.

Metrics for Pascal and Miranda: The Halstead and McCabe metrics for the imperative programming language Pascal have been used as described by Conte (Conte et al., 1986). All variables and constants are counted as operands. The operators are the arithmetic operators, relational operators, boolean operators, procedure and function counts, and multiple entities BEGIN END, IF THEN, IF THEN ELSE, WHILE DO, FOR DO, REPEAT UNTIL, CASE END, RECORD END, ARRAY OF, SET OF. The decision count for the cyclomatic complexity number is based on the occurrence of the symbols WHILE, FOR, REPEAT, IF, OTHERWISE, AND, OR, PROCEDURE, FUNCTION, PROGRAM, CASE and commas in the CASE statement.

We have developed the Halstead and McCabe metrics for the functional programming language Miranda. (Similar metrics have been established by Samson, Dugard, Nevill, Oldfield & Smith (1989), for the languages Hope and OBJ.) We consider constants and all identifiers that are not operators as operands. The operators are the standard operators (including the list operators for list construction ':', list concatenation '++', list difference '--' and selection from a list '!'), the function name in the right hand side (RHS) of function definitions, parameters in a compound definition, delimiters in expressions ('[ ]', '( )','..',';',','), and in function definitions the symbols 'otherwise' and 'where'.

The number of decisions has to incorporate the use of patterns in arguments of function definitions, in compound definitions and in list comprehensions (i.e., expressions from the Zermelo

Frankel set theory). A function definition consists of a left-hand side (LHS), the symbol '=' and the right-hand side (RHS).

The cyclomatic complexity number of a script is equal to 1 + the sum of cyclomatic complexity numbers of each function definition. The cyclomatic complexity number of a function definition is equal to the sum of the pattern complexity of the LHS, the number of guards in the RHS, the number of logical operators in the RHS, the number of filters in a list comprehension in the RHS, and the pattern complexity in a list comprehension in the RHS. The pattern complexity is equal to the number of identifiers in the pattern, minus the number of unique identifiers in the pattern, plus the number of arguments that are not identifiers.

We illustrate the determination of these metric values for a small functional program. Suppose we want to calculate a list with all numbers greater than 5 of a given list with numbers, e.g., *filter (>5) [1,7,2,6]*. In Miranda the definition of *filter* reads as follows, where *[]* denotes the empty list, and *(x:xs)* denotes an item *x* at the head of a list of items *xs*.

filter p [] = []
filter p (x:xs)= x : filter p xs , p x
= filter p xs , otherwise

In this example, the Halstead metric values are $\eta_1 = 7, N_1 = 14, \eta_2 = 3$ and $N_2 = 11$. These values have been derived from the following < operator , frequency > tuples: < '( )' , 1 >, < '[ ]' , 2 >, < ',' , 1 >, < 'otherwise' , 1 >, < ':' , 2 >, < '=' , 3 >, < 'filter' , 4 >. The < operand , frequency > tuples are: < 'x ', 3 >, < 'xs' , 3 >, < 'p' , 5 >. We assume $\eta_2^* = \eta_2 = 3$. The other Halstead quantities can be derived from these basic values.

The calculation of the McCabe metric value proceeds as follows. The number of arguments that are not identifiers is 2, resulting in a pattern complexity of 2. The number of guards is 1. The cyclomatic complexity number is 1 + (1 + 2) = 4.

Automated measurement: The metrics of Halstead and McCabe are based on the lexical and syntactical analysis of the program code. It is possible to use standard tools like a scanner and parser to automate the measurement of these metric values. In our study we used the Cornell Synthesizer Generator (Reps & Teitelbaum, 1989; Robbers, 1990). A schematic view of the analyser is given in Figure 1. (A similar metric analyser has been developed by Henry & Goff, 1989.)

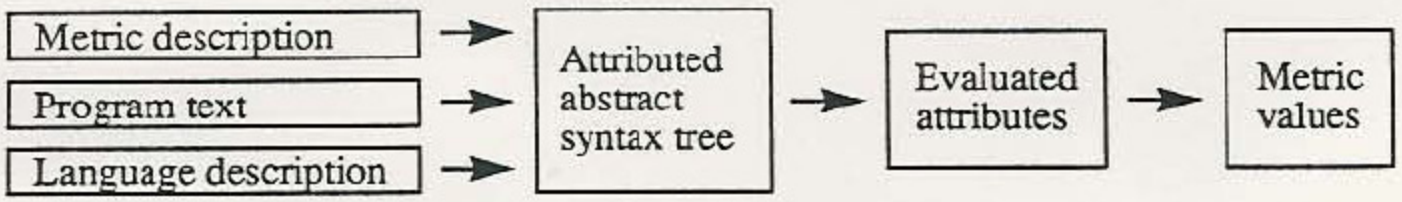

Figure 1: Schematic view of the software metric analyser.

From a syntactical correct program text an abstract syntax tree is derived using the parse rules and a language description in the Extended Backus Naur Form. The metric values are calculated by means of attribute rules, provided in the metric description. For each node of the tree we can determine the complexity from the lower parts in the tree. The attribute rules have been developed for Pascal and Miranda in order to calculate the Halstead and McCabe metrics, and can be extended to other metrics. The output of the generator can be used in an auxiliary program to calculate the actual metric values.

Case study

In the case study we explored the application of the syntactic complexity metrics described in the previous sections to programs in the functional programming language Miranda, and we established the correlation between these metrics and the readability of the programs. We used programs written in the language Pascal for comparison. The programs were taken from two groups of first year undergraduate students in Computer Science, after a programming course of one term. The (first) experimental group in Miranda and the second (control) group in Pascal. The students were asked to extend an existing program in the respective language. We compared their modifications of this reference program. The reference program carried out the conversion of a string representing a Roman number, e.g., MCLVII, to the corresponding decimal number, 1157. The extension of the program should allow the usual abbreviations in the input string, such as XL instead of XXXX. A conversion of the string MCXLVII should result in the decimal value 1147.

The readability of programs was assessed by 11 experts, all lecturers in Computer Science. They established a rank-order of readability for 9 modifications of the reference program in Pascal from the control group, and 8 modifications in Miranda from the experimental group. The agreement between the rankings is given by Kendall's coefficient of concordance W (Lindeman, Merenda & Gold, 1980). The results are given in Table 1. From these rankings we calculated the average expert rank of the programs in each group.

Table 1: Correlation between expert rankings of readability for Pascal and Miranda programs.

Language	Number of programs	Correlation	Significance
Pascal	9	$W = .74$	$\chi^2(8) = 65^*$
Miranda	8	$W = .50$	$\chi^2(7) = 39^*$

Notes. The correlation W is Kendall's coefficient of concordance.
The significance is tested by means of the χ^2-test. $^*p < .001$.

Using the metric analyser, we determined the values of the software metrics for the reference programs and the modified programs in both languages. The ratings for the modified programs on Halstead's effort E and McCabe's cyclomatic complexity number were converted to rankings, or an average rank in the case of equal ratings. The correlation between these rankings and the average expert rank was calculated with Spearman's rank-order correlation coefficient r_s. The values of these correlations are given in Table 2. We will evaluate the results of this case study in the following section.

Discussion

The metrics of Halstead and McCabe for the static syntactic complexity of programs in the functional programming language Miranda have been defined and the measurement has been auto-

mated. In the case study we compared the application of these metrics to programs in the functional language Miranda and the imperative language Pascal. A ranking of the expert assessment on readability was taken as a measure of the psychological complexity. From the values of the correlation between these expert rankings, we conclude that there is fair agreement between experts about the ranking of programs in Pascal (W= 0.74), but the agreement is low for programs in Miranda (W= 0.50). This could mean that there is not as much an accepted standard on readability for Miranda programs as for Pascal programs.

Table 2: Correlation between rankings of complexities for Pascal and Miranda programs.

Complexity 1	Complexity 2	Correlation	Significance
	Extended Pascal programs (N = 9)		
Effort	Cycl Compl Number	$r_s = .61$	t(7) = 2.01 ***
Av Expert Rank	Effort	$r_s = .90$	t(7) = 5.46 ****
Av Expert Rank	Cycl Compl Number	$r_s = .58$	t(7) = 1.88 ***
	Extended Miranda programs (N = 8)		
Effort	Cycl Compl Number	$r_s = .89$	t(6) = 4.78 ****
Av Expert Rank	Effort	$r_s = .38$	t(6) = 1.01 *
Av Expert Rank	Cycl Compl Number	$r_s = .56$	t(6) = 1.66 **

Notes. The correlation r_s is Spearman's rank-order correlation coefficient.
The significance is tested by means of the Student t-test.
*p<.20. **p<.10. ***p<.05. ****p<.01.

The correlation between the effort rank and the rank on the cyclomatic complexity number for the programs in Pascal is $r_s = 0.61$, and for Miranda $r_s = 0.89$. Both correlations are significant. The high value for Miranda indicates a consistent measurement of the syntactic complexity metrics as developed in this study.

The correlation between the average expert rank and the cyclomatic complexity order for Pascal is $r_s = 0.58$, and for Miranda $r_s = 0.56$. The correlation between the average expert rank and the effort order for Pascal is $r_s = 0.90$, and for Miranda $r_s = 0.38$. Both correlations for Pascal are high and significant, which is in agreement with the literature. The two correlations for Miranda are not significant. Obviously, this could be caused by the small number of programs used in this study. There are two reasons which could explain the low correlations for Miranda. Firstly, the value on the coefficient of concordance, indicating the agreement between experts on readability, for Miranda programs is low (see above). The second reason can be found in a more general argument given by Halstead (1977). He pointed to the dual role of the level of implementation of a program with respect to the understandability. For an expert the understandability is proportional to the program level, whereas for a novice the understandability is inversely proportional to the program level. In other linguistic studies, it was concluded that experienced

writers cannot reliably predict the readability to novices of text in natural languages (Baker, Atwood & Duffy, 1988).

Further study of software metrics for functional programming languages is required. There is a need to differentiate between metric values for various operators. Especially the use of higher-order functions can result in concise programs, and the count of such an operator should have a different contribution to the complexity than a simple arithmetic operator. It is necessary to establish the psychological complexity of each elementary language construct and the complexity of the compositions of these constructs in programs. This could be carried out in an axiomatic approach as outlined by Fenton (1991), along with contributions from measurement theory to software measurement (Weyuker, 1988; Zuse, 1991).

The complexity of the program text can be analysed at different linguistic levels: at lexical level, i.e., the vocabulary used in the program, at syntactical level, i.e., the linguistic structures in the program, and at the level of the program text as a whole, including the composition (sequencing and nesting) of linguistic structures. In this study we used the parsing of the program text by the computer as a model for the parsing by the human reader. The analysis should be based on a psychologically plausible parsing model, which is part of a cognitive model for the comprehension process. The Halstead metrics and McCabe metrics focus on the lexical and syntactic levels, and may not represent the most adequate levels for measurement of program comprehensibility (Card & Glass, 1990). We also have to incorporate the text level of analysis, as we see in discussions on readability and linguistic complexity of natural language texts (Frazier, 1988). The processing of program text has to be formulated in terms of comprehension of chunks of code and programming plans (Davies, 1989; Green & Borning, 1990), and the problem-solving capacities of programmers (Curtis et al., 1979). Furthermore, if we want to compare the understandability and the expressive power of programming languages we should weigh up metrics for programs and metrics for languages (Sammet, 1981).

Before metrics can be used in teaching programming as feedback to students on their programs, further development of software metrics is necessary together with the development of criteria to assess the readability of programs written in a functional programming language.

Acknowledgments

I would like to thank Jeroen van Merriënboer and Herman Koppelman for their comments on an earlier version of this paper; Mieke Massink for setting up the experiment with two programming groups; Robert Couweleers and Diederik de Rooij for the implementation of the metric analyser and the collection of data. I thank Gerard Kempen for bringing to my attention recent work by Green and Borning.

References

Baker, E.L., Atwood, N.K., & Duffy, T.M. (1988). Cognitive approaches to assessing readability of text. In: A. Davison & G.M. Green (Eds). *Linguistic Complexity and Text Comprehension: Readability Issues Reconsidered*, 55-84. Hillsdale, NJ: Lawrence Erlbaum Associates.

Bailes, P.A., & Salzman, E.J. (1989). A proposal for a bachelors degree program in software engineering. *Software Engineering Education. Lecture Notes in Computer Science, 376*, 90-108. Berlin: Springer-Verlag.

Bailey, R. (1990). *Functional Programming with HOPE*. Chichester, U.K.: Ellis Horwood.

Bird, R., & Wadler, P. (1988). *Introduction to Functional Programming*. New York: Prentice Hall.

Boehm, B.W., Brown, J.R., Kaspar, H., Lipow, M., & Merritt, M.J. (1978). *Characteristics of Software Quality*. Amsterdam: North-Holland.

Card, D.N., & Glass, G.L. (1990). *Measuring Software Design Quality*. Englewood Cliffs, NJ: Prentice Hall.

Conte, S.D., Dunsmore, H.E., & Shen, V.Y. (1986). *Software Engineering Metrics and Models*. Menlo Park, CA: Benjamin/Cummings.

Coulter, N.S. (1983). Software science and cognitive psychology. *IEEE Trans. Software Engineering, SE-9(2)*, 166-171.

Curtis, B. (1981). The measurement of software quality and complexity. In: A. Perlis, F. Sayward, & M. Shaw. (Eds). *Software Metrics: An Analysis and Evaluation*, 203-223. Cambridge, Mass: MIT Press.

Curtis, B., Sheppard, S.B., Milliman, P.M.A., & Love, T. (1979). Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Trans. Software Engineering, SE-5(2)*, 96-104.

Davies, S.P. (1989). Skill levels and strategic differences in plan comprehension and implementation in programming. In: A. Sutcliffe & L. Macaulay (Eds). *Peoples and Computers V*, 487-502. Cambridge: Cambridge University Press.

Fenton, N.E. (1991). *Software Metrics: A rigorous approach*. London: Chapman & Hall.

Fleck, A.C. (1990). A case study comparison of four declarative programming languages. *Software-Practice and Experience, 20(1)*, 49-65.

Frazier, L. (1988). The study of linguistic complexity. In: A. Davison & G.M. Green (Eds). *Linguistic Complexity and Text Comprehension: Readability Issues Reconsidered*, 193-221. Hillsdale, NJ: Lawrence Erlbaum Associates.

Green, T.R.G., & Borning, A. (1990). The generalized unification parser: Modelling the parsing of notations. In: D. Diaper, G. Cockton, D. Gilmore, & B. Schackel. (Eds). *Human-Computer Interaction-INTERACT '90*, 951-957. Amsterdam: North-Holland.

Halstead, M.H. (1977). *Elements of Software Science*. New York: Elsevier.

Henry, S., & Goff, R. (1989). Complexity measurement of a graphical programming language. *Software-Practice and Experience, 19(11)*, 1065-1088.

Joosten, S.M.M., & Berg, K.G. van den (1990). Can computer programming be based on functional programming. *Memoranda Informatica*, 90-46. Enschede, University of Twente.

Khalil, O.E., & Clark, J.D. (1989). The influence of programmer's cognitive complexity on program comprehension and modification. *Int. Journal Man-Machine Studies, 31*, 219-236.

Lindeman, R.H., Merenda, P.F., & Gold, R.Z. (1980). *Introduction to Bivariate and Multivariate Analysis*. Glenview, IL: Scott, Foresman and Company.

McCabe, TJ. (1976). A complexity measure. *IEEE Trans. Software Engineering, SE-2(4)*, 308-320.

Petre, M., & Winder, R. (1990). On languages, models and programming styles. *The Computer Journal, 33(2)*, 173-180.

Reps, T.W., & Teitelbaum, T. (1989). *The Synthesizer Generator: A System for Constructing Language-based Editors*. New York: Springer-Verlag.

Robbers, R.M.R. (1990). Software metric analysers based on attribute grammars. The metrics of Halstead and McCabe for Pascal and Miranda programs. *Memoranda Informatica*, 88-90. Enschede, University of Twente.

Sammet, J.E. (1981). High level language metrics. In: A. Perlis, F. Sayward & M. Shaw (Eds). *Software Metrics: An Analysis and Evaluation*, 131-141. Cambridge, MA: MIT Press.

Samson, W.B., Dugard, P.I., Nevill, D.G., Oldfield, P.E., & Smith, A.W. (1989). The relationship between specification and implementation metrics. In: B.A. Kitchenham & B. Littlewood (Eds). *Measurement for Software Control and Assurance*, 335-384. London: Elsevier.

Sanders, P. (1989). An evaluation of functional programming for the commercial environment. *Br. Telecom. Technol. Journal, 7(3)*, 25-33.

Shen, V.Y., Conte, S.D., & Dunsmore, H.E. (1983). Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Trans. Software Engineering, SE-9(2)*, 155-165.

Springer, G., & Friedman, D.P. (1990). *Scheme and the Art of Programming*. Cambridge, MA: MIT / New York: McGraw-Hill.

Turner, D. (1986). An overview of Miranda. *Sigplan Notices, 21(12)*, 158-166.

Weyuker, E.J. (1988). Evaluating software complexity measures. *IEEE Trans. Software Engineering, SE-14(9)*, 1357-1365.

Zuse, H. (1991). *Software Complexity: Measures and Methods*. Berlin: De Gruyter.