Validation of Structure Metrics: A Case Study

K. G. van den Berg, P. M. van den Broek & G. M. van Petersen

University of Twente, Faculty of Computer Science, P.O.Box 217, 7500 AE Enschede, the Netherlands, e-mail: vdberg@cs.utwente.nl

Abstract

A framework for the validation of axiomatic structure metrics is presented. In a case study, the comprehensibility of type expressions in the functional programming language Miranda[1] has been investigated. A structure metric for the comprehensibility of type expressions has been developed together with internal and external axioms. This structure metric has been validated experimentally. The calibrated metric function results in a good prediction of the comprehensibility.

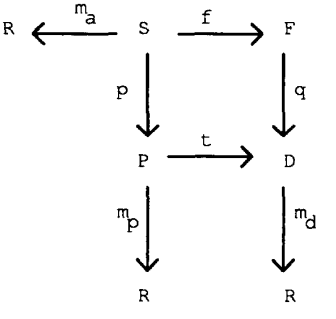
1. Introduction

Software metrics are used to quantify objectively attributes of software entities [4]. Three types of entities can be distinguished: products, processes and resources. Furthermore, there are two types of attributes: internal attributes and external attributes. The latter not only depends on the software entity, but also on other entities. Examples of internal attributes of software products are size and structure; maintainability and reusability are examples of external attributes. Structure metrics aim to quantify the internal structure of the product. A general theory of structure metrics is provided by [3]. Structure metrics are based on the compositionality principle.

```
If
   S,S1,...,Sn: System
   C: System × ... × System → System
   m: System → R
then there exists a function f_C
   f_C: R × ... × R → System
such that
   S = C(S_1,...,S_n) ⇒
            m(S) = f_C((m(S_1),...,m(S_n))
where
   R is a real number
   C is a system constructor
   m is a measure of attribute A
```

This principle asserts that the property of a system can be derived from the properties of its constituent components without knowledge of the interior structure of those components. Interaction between properties of components is excluded.

A scheme for the measurement of a software system is given in figure 1 (cf. [11]). Structure metrics have been developed for computer programs in imperative programming languages.




```
S   = system          R   = real number
f   = front-end       F   = flowgraph
q   = decomposition   D   = decomposition tree
p   = parser          P   = parse tree
t   = tree transform  m_a = metric on S
m_p = metric on P     m_d = metric on D
```

Figure 1. Scheme for measurement of software

The control flow in a program S is modelled in a flowgraph F by a function f. By defining two constructors on flowgraphs, sequencing and nesting, a decomposition algorithm q yields a unique decomposition tree D. Consequently, a metric function m_d can be defined on this tree structure resulting in a number R. Moreover, there is a non-structural measurement m_a of the system with respect to attribute A. The order in the result of this m_a on systems should correspond to the order on systems from the composition

[1] Miranda is a trade mark of Research Software Limited

0-8186-3740-4/93 $3.00 © 1993 IEEE

of functions $m_d \cdot q \cdot f$. A tool supporting this analysis is Qualms [1]. There are several front-ends for the modelling of programs in flowgraphs, i.e. the function f.

An alternative to this approach is using a grammar to define systems. A parse tree P is the result of the parse p of system S. Again, a metric function m_p can be defined on this tree structure. The order in the result of this m_a on systems should correspond to the order on systems from the composition of functions $m_p \cdot p$. The existence of a function t, which transforms a parse tree to a decomposition tree has to be investigated.

Grammars are used in complexity rankings of programs ([18], [15]). The use of grammars is similar to the approach with algebraic structures as the base for compositional analysis [21]. Algebraic specification has been used in the validation of software metrics [14]. Attribute grammars have been used in software metrics [2]. Structure metrics have been defined with attribute grammars [19].

In the case study, the investigated software products are type expressions in the functional programming language Miranda. Type expressions and a structure metric are described in paragraph 3. The external attribute of these products is the comprehensibility to a human reader. The measurement of the comprehensibility will be described in paragraph 4. The general framework for the experimental validation is described in the following section.

2. A framework for validation

A scheme of the framework for the experimental validation of structure metrics is displayed in figure 2. Some model, a flowgraph or a grammar, will be used to model the structure of the software product and results in a tree structure. The internal axioms provide the definition of a structure metric: this reflects the compositionality. The external axioms state the properties of the software entities and give the hypothetical order of these entities with respect to the external attribute.

The validation of the metric function consists of the following six steps:

a. The function satisfies the internal axioms: consequently, the function is a structure metric.

b. The function satisfies the external axioms: it provides a consistent measure with respect to the external attribute. This results in conditions on coefficients in the metric function.

c. The external axioms hold in practice: the actual order on software products, with respect to the external attribute, corresponds to the hypothetical order expressed in the external axioms.

d. The function is calibrated: the coefficients are given actual values, determined from a non-structural measurement of the software products with respect to the external attribute.

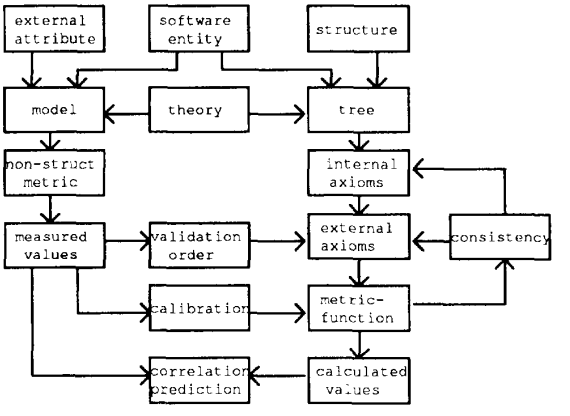


Figure 2. Framework for the validation of structure metrics for an external attribute of a software entity

e. The calibrated function is used for the rank order: the rank order correlation between measured and calculated values is determined.

f. The calibrated function is used for prediction (in stochastic sense): the efficiency of the prediction of actual values from calculated values is determined.

In the following section, the software entity in the case study - type expressions in the functional programming language Miranda - will be introduced. In addition, a structure metric for type expressions will be described.

3. Structure metrics of type expressions

In this paragraph, a subset of type expressions in Miranda, will be described. A grammar for this subset will be presented, followed by some alternative grammars. The internal axioms for a structure metric for type expressions will be given, and subsequently, the external axioms and the metric function itself.

3.1. Type expressions

Many programming languages provide some kind of typing system. In Modula-2, the type of variables has to be declared. The heading of a procedure declaration must contain the types of the parameters and the result.

E.g., the function procedure

```
PROCEDURE Digit (K: CHAR): BOOLEAN
```

In the functional programming language Miranda it is optional to the programmer to provide the type of a function. The type-checker derives the type and compares this with the given type. The syntax of type expressions will be illustrated with some examples in table 1. There are simple standard types, such as `char`, `bool` and `num`. The function constructor is denoted with an arrow `->`. The function `digit` has the type:

```
digit :: char -> bool
```

Furthermore, there are type variables [17], in Miranda denoted with one or more stars. Structured standard types are lists, denoted with square brackets, and tuples, denoted with round brackets. In each example, the type of the function is given and an informal description. After the question mark prompt, a function application is given with its result on the next line.

```
digit :: char -> bool
The function digit returns True if the argument
is a digit and otherwise False

? digit '5'
True

head :: [*] -> *
The function head returns the first element of a
given list

? head [2,4,7,4]
2

first :: (*,**) -> *
The function first returns the first component
of a given 2-tuple

?first ('5',True)
'5'

split :: (*-> bool) -> [*] -> ([*],[*])
The function split returns, given a predicate
(boolean function) and a list, a tuple with the
first component the list with elements
satisfying the predicate and the second
component the list with elements not satisfying
the predicate

? split even [2,4,7,4]
([2,4,4],[7])
```

Table 1. Examples of type expressions with function applications in Miranda

3.2. A grammar for type expressions

Structure metrics for Miranda type expressions are derived from a grammar. The grammar for a subset of type expressions is given below: here, a Miranda data structure is used to model this grammar.

```
typeexp ::= Num | Bool | Char | Var num |
   L typeexp | T [typeexp] | F typeexp typeexp
```

The first line of the grammar gives the rules for the prime structures and the second line gives the rules for the three constructors in type expressions: the list constructor, the tuple constructor and the function constructor. The type

```
(* -> bool) -> [*] -> ([*],[*])
```

of the function `split` from table 1 can be parsed with this grammar, resulting in:

```
(F
   (F (Var 1) Bool)
   (F
      (L (Var 1))
      (T [L (Var 1)), (L (Var 1)])))
```

The parse tree or derivation tree of the function `split` is given in figure 3.

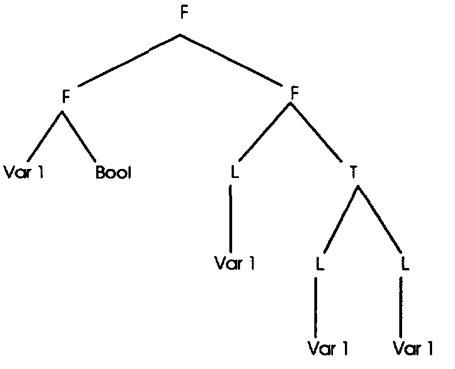


Figure 3. Derivation tree of the type expression of the function *split*

3.3. Alternative grammars

The grammar described above is based on the right associativity of the function arrow. The type of the function `split` can be structured as a function with one argument (the predicate `(* -> bool)`) and with as result a function with the type `([*] -> ([*],[*])`. This approach is named currying [17]. In other words, the type of `split` has been structured as follows:

```
(* -> bool) -> ([*] -> ([*],[*]))
```

There are two alternatives to this grammar. First, the type of the function can be structured as a function with one argument of the product type `((* -> bool) × [*])` and with a result of type `([*],[*])`. The grammar for the function constructor in this case

contains

```
F [typeexp] typeexp
```

The second alternative is obtained when the type of the function is structured in a similar way as the tuple: each function arrow separates types in the function constructor, in the same way as the comma separates the types of the components in a tuple.

The rule for the function constructor in this case is

```
F [typeexp]
```

Clearly, the derivation tree, and derived properties such as depth, depends on the chosen grammar. The ultimate choice of the grammar is determined by the psychological plausibility of the parsing model with respect to comprehension, and not by the actual parsing of the compiler. This approach has been used in the parsing of natural language sentences (Derivational Theory of Complexity, [5]). New theories on the comprehension processes for natural languages point to shortcomings of this approach [10]. One might expect interaction between properties of constituent components. However, this theory could be adequate for the human parsing of simple expressions in formal languages (cf. [6]).

In the further validation study, the first grammar - based on the right associativity of the function arrow - has been used.

3.4. The internal axioms

A function m is a structure metric if it is defined according to the compositionality principle. For type expressions, a structure metric should satisfy the conditions listed in table 2. These conditions are called the internal axioms. The first four axioms refer to the prime structures and the constants c_i denote any number. The final three axioms refer to the constructors of type expressions.

m(Num)	$= c_N$
m(Char)	$= c_C$
m(Bool)	$= c_B$
m(Var n)	$= c_V(n)$
m(L t)	$= f_L(m(t))$
m(F t_1 t_2)	$= f_F(m(t_1),m(t_2))$
m(T[t_1,...,t_n])	$= f_T(m(t_1),...,m(t_n))$

Table 2. Internal axioms for the structure metric of type expressions

3.5. The external axioms

The order on software entities with respect to a certain attribute should be reflected in the values obtained by the metric functions. This order is described in an extension of the set of axioms, as has been done for flowgraphs [4]. These additional axioms are the hypotheses that will have to be tested. They will be referred to as the external axioms. The software entities in this case study are the Miranda type expressions, whereas the external attribute is the comprehensibility of these expressions. Let t and t_k ,... be type expressions. There are many possible hypotheses about the intuitive order, as will be seen below:

1. An order between the prime structures, e.g.:

```
m(Var n) > m(Bool) > m(Char) > m(Num)
```

2. An order between type expression with a constructor and with its components, e.g.:

2.1 An order on (L t) and t

```
m(L t)      > m(t)
```

2.2 An order on (F t_1 t_2) and t_1 and t_2

```
m(F t1 t2)  > m(t1)
m(F t1 t2)  > m(t2)
m(F t1 t2)  > max(m(t1),m(t2))
m(F t1 t2)  > m(t1) + m(t2)
```

2.3 An order on T[t_1,...,t_n] and t_1 ... t_2

There are similar hypothetical orders as on F, but generalised for the number of components. Some of these possibilities have been illustrated in figure 4.

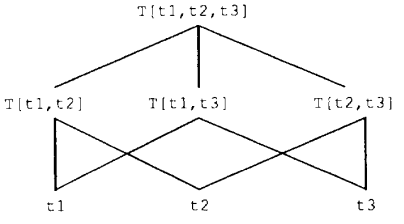


Figure 4. Order of type expressions with the tuple constructor

3. An order between type expressions with the same constructor, e.g.:

```
m(F t1 t2)              = m(F t2 t1)
m(T[t1,...,tn+1])       > m(T[t1,...,tn])
m(T[t1,...,tn])         = m(T[ti,...,tj]),
  where [ti,...,tj] ∈ perms[t1,...,tn]
```

4. An order between types with different constructors

$m(F\ t_1\ t_2) > m(L\ t_i),\ i=1,2$
$m(T[t_1,t_2]) > m(L\ t_i),\ i=1,2$
$m(F\ t_1\ t_2) > m(T[t_1,t_2])$

The external axioms on the comprehensibility as used in the case study are listed in table 3. These hypotheses have been validated experimentally. This will be described in paragraph 4.

1. $m(L\ t)$	$> m(t)$
2. $m(T[t_1,\ldots,t_n])$	$> \max(m(t_1),\ldots,m(t_n))$
3. $m(T[t_1,\ldots,t_n])$	$= m(T(perm[t_1,\ldots,t_n]))$
4. $m(Ft_1\ t_2)$	$= m(F\ t_2\ t_1)$
5. $m(T[t_1,\ldots,t_{n+1}])$	$> m(T[t_1,\ldots,t_n])$
6. $m(T[t_1,\ldots,t_n])$	$> m(L\ t_i),\ i=1,\ldots,n$
7. $m(F\ t_1\ t_2)$	$> m(T[t_2,t_1])$

Table 3. External axioms for the structure metric with respect to the comprehensibility of type expressions

3.6. The metric function

There are many candidates for the metric function on the tree structure, that has been obtained so far; e.g., there are the sum and product VINAP-measures [4]. In the Qualms system, many more metrics are available. From a compositional theory for the attribute, the actual choice can be made. However, the final choice will be determined by the performance of the metric function in a prediction system. For the type expressions in the case study, a simple sum metric has been chosen, as listed in table 4.

$m(Num)$	$= c_N$
$m(Char)$	$= c_C$
$m(Bool)$	$= c_B$
$m(Var\ n)$	$= c_V(n)$
$m(L\ t)$	$= c_L + m(t)$
$m(F\ t_1\ t_2)$	$= c_F + m(t_1) + m(t_2)$
$m(T[t_1,\ldots,t_n])$	$= c_T + m(t_1) + \ldots + m(t_n)$

Table 4. Metric function m for the structure of type expressions

This function m is consistent with the given internal axioms (table 2). The constants c_L, c_F, c_T, c_N, c_C, c_B and $c_V(n)$ should fulfil certain conditions, which can be derived from the external axioms (table 3) and the function definition (table 4), in order that the function is in agreement with the external axioms.

In the following paragraph, the experimental validation of this structure metric for the comprehensibility of type expressions will be described. The actual values for the constants in the metric function m will be established.

4. Validation

The validation of structure metrics consists of six steps, as outlined in paragraph 2. The experimental validation of a structure metric for the comprehensibility of type expressions will be described now. The first two steps of the validation - the proof that the metric function satisfies the internal and external axioms - have been accomplished in the previous paragraph. The next four steps of the validation have to be carried out experimentally:

c. The external axioms hold in practice
d. The function is calibrated
e. The calibrated function is used for the rank order correlation
f. The calibrated function is used for prediction

Steps c and d are established in experiment 1 and steps e and f are verified in experiment 2. A detailed account of these experiments is given in [12]. For the (non-structural) measurement of comprehensibility of programs, there are several techniques known from literature [13]:

1. answering (multiple choice) questions
2. filling in blanks
3. writing a program for a given input and output
4. writing fitting input and output for a given program
5. modifying an existing program
6. localising errors in a program

In this case study, a variant of the third technique has been chosen. A type expression will be shown to the subject. He or she is requested to write a function, that will result in exactly this type when offered to the Miranda type checker. The time is measured between the moment that the type expression is shown to the subject until the answer is completed by the subject. For example, if the following type expression is shown

```
f  :: (num -> char) -> char -> bool
```

then the following definition will be a correct answer:

```
f g 'a' = True, if g 5 = 'b'
```

It is not required that the function itself has any sensible meaning; just the given type must agree exactly with the type of the function obtained by the type checker.

4.1. Method

The subjects in the experiments are 16 first year

students in Computer Science at the University Twente. During one term, they followed a course in Functional Programming with Miranda [9]. They volunteered to the experiments and were randomly distributed over the two.

The material consisted of 40 questions with type expressions offered to the subjects. In each question the subject has to answer with a definition that matches with the given type expression.

4.2. Procedure

The questions are offered to the subjects on a system in the computer science laboratory (SUN workstations). The subjects know this system from their practical assignments in the programming course. First, there is a short introduction on how to answer the questions, and subsequently, two questions are presented for trial. With the standard editor (Vi) the subjects type their conceived answer. The time, elapsed between showing the question and leaving the editor, is measured automatically by the system. A countered balanced design is chosen in this experiment. All subjects get the same questions, but they are offered in a random order different to each subject.

4.3. Results

The hypothetical order on type expressions has been expressed in the external axioms. Each axiom has a left hand side (LHS) and a right hand side (RHS). The questions are assigned to the LHS and RHS of the axioms. In this way, questions can be used more than once. This approach is someway similar to the idea of atomic modifications in [20].

For example, axiom 1 states that `m(L t) > m(t)`. A question pair is for the LHS `[char -> bool]` and for the RHS `char -> bool`.

A within subject design is chosen. Per axiom and per student the average time is calculated for LHS-questions and RHS-questions. Only questions that belong to an axiom of which both sides are answered correctly are taken into account. Type writing errors in the answers have been corrected. The measured time is adjusted for an individual offset-time: the time for a subject to go with the cursor to the answer frame and leaving the editor, without giving an answer. Extreme values are discarded (in which the difference between the averaged LHS and RHS times for an axiom differs more than three times the standard deviation from the arithmetic mean). The differences between the LHS and RHS appear to have a normal distribution. The average time is calculated for each side and each axiom from the averages of all students. The results are given in table 5.

The significance of the difference between the LHS and the RHS is calculated with the Fischer t test (which applies to differences between correlated pairs of means [7]). The degree of freedom is determined by the number of correct answer pairs (and not merely by the number of subjects). The results are shown in table 5.

	axiom	t LHS (sec)	t RHS (sec)	Fischer-t(n)
1	LHS>RHS	19.0	08.0	t(26)=+7.09*
2	LHS>RHS	21.6	10.6	t(27)=+5.08*
3	LHS=RHS	33.8	29.7	t(15)=+0.70
4	LHS=RHS	15.2	20.7	t(22)=-3.12*
5	LHS>RHS	25.6	20.5	t(20)=+1.08
6	LHS>RHS	24.6	12.7	t(25)=+4.08*
7	LHS>RHS	19.7	12.7	t(17)=+2.03

n = # degrees of freedom, * = p < .05

Table 5. Results of the validation of the external axioms with respect to the comprehensibility of type expressions

The measured values of the times for the good answers in the first experiment are also used for the calculation of the values of the coefficients in the metric function. The questions are grouped now per constructor. For example, the calculation of the coefficient c_T from the equation:

```
m(T[Num, Bool, F Char Bool])
  = cT + m(Num) + m(Bool) + m(F Char Bool))
```

The time measured for the type expression left is 48 seconds and for the type expressions right is measured 5.0, 7.0 and 27.5 seconds respectively. From these values, c_T has been calculated and averaged over the other values obtained for c_T. In table 6 the average values for the coefficients are given.

cL	cT	cF	cC	cN	cB	cV
10	6	7	4	5	7	19

Table 6. Values for the primes and the constants in the metric function (seconds)

In the second experiment, a new set of questions is offered to the second group of subjects. The average time is calculated from each good answer and, based on these values, the rank order of type expressions has been established. Moreover, with the calibrated metric function from the first experiment, the rank order of the same type expressions has been calculated. The correlation between both orders has been determined according to Spearman [7]. The rank order correlation coefficient is 0.59. (Pearson's coefficient can not be used because the scores have been obtained in dependent pairs).

On the basis of the calculated value of the

comprehensibility with the calibrated metric function, a prediction can be made of the actual comprehensibility (as would have been obtained by measurement). The forecasting efficiency [7] is 19%; i.e. a reduction in variance of the predicted comprehensibility is achieved by using the calculated metric value.

5. Discussion

From the values in table 5, it appears that for axioms 1, 2 and 6 there is a significant difference between the LHS and RHS, according the hypothesised order. For axiom 5 and 7, no significant difference has been found. In case of axiom 5, a possible cause of this fact could be that the expansion of a tuple with a component only gives a small, and therefore a difficult to measure, effect. For axiom 7, the reason of the small difference is not clear. Axioms 3 and 4 have to be treated separately. It seamed to be reasonable to include equalities in external axioms. However, equalities can not be validated experimentally in the way described before. Therefore, nothing can be concluded from the results for these two axioms.

It has been expected that the comprehensibility of a function is more difficult than of a tuple with the same components. Table 6 shows that the actual difference is small (but significant). The value for the constant c_V for type variables is remarkably high.

The rank order correlation coefficient, for the calculated and measured values, is reasonably high. However, this coefficient is as high if the metric function returns the numbers of nodes and leaves in the decomposition tree. This case can be seen as the Halstead measure for the length of a 'program' [8]. The nodes are the total number of operators and the leaves the total number of operands. The so defined measure is a structure metric. It has not been checked whether this length metric satisfies all internal axioms. A high correlation coefficient is found as well if the calculated rank order is based simply on the size of the type expressions, i.e. the number of characters.

The forecasting efficiency is reasonably high. An even higher value (53%) is obtained when the values are grouped (16 groups). This leads to a considerable reduction in the variance of the prediction of the comprehensibility from the calculated metric value.

6. Conclusion

In this paper a framework has been presented for the experimental validation of structure metrics. In a case study, a structure metric and its validation for the comprehensibility of type expressions in Miranda has been studied within this framework. No hard conclusions can be drawn about the absolute values obtained in the experiments. There is need for additional experiments. The validation for the alternative grammars of type expressions has to be carried out. Metric functions, which incorporate the depth of nesting, have to be investigated. The influence of type expressions for standard functions has to be included (e.g. the type expression `(* -> *)` will be recognised as belonging to the standard identity function `id`). The set of primes has to be extended (e.g. a list of `char` will be comprehended as a string). The experiments have to be extended to include the whole Miranda type language (e.g. type synonyms, recursive types and abstract data types should be included). The effect of multiple occurrences of types in type expressions, which cannot be accounted for with compositionality, has to be investigated.

There are some general conclusions from this case study. The use of grammars, as an alternative to flowgraphs in the modelling of software in a tree structure, has been shown. The role of the external axioms, to express the hypothetical order on the software entities with respect to the external attribute, has been emphasised. In a prediction system based on structure metrics, there has to be a theory of compositionality for the external attribute. The experimental validation of the hypothetical order and the calibration of the metric function both require a large amount of experimental data on the software entities. Statistical analysis is needed to establish the actual order on these software entities and for the calculation of quantities, such as the rank order correlation coefficient and the forecasting efficiency.

Acknowledgement

The authors would like to thank Hendrik Muijs for the modelling of the Miranda type expressions as part of his M.Sc. thesis, and Jeroen van Merriënboer for valuable comments on the experimental set up and the statistical analysis.

References

[1] Bache, R. & Leelasena, L. (1990), *Qualms, A Tool for Control Flow Analysis and Measurement*. CSSE, South Bank Polytechnic, London.

[2] Berg, K.G. van den (1992), Syntactic Complexity Metrics and the Readability of Programs in a Functional Computer Language, In: F.L. Engel, et al (Eds), *Cognitive Modelling and Interactive Environments in Language Learning*, NATO Advanced Science Institute Series, Berlin: Springer, 199-206.

[3] Fenton, N. E. & Kaposi, A. A. (1989), An Engineering Theory of Structure and Measurement. In: B. A. Kitchenham & B. Littlewood (Eds.). *Measurement for Software Control and Assurance*, London: Elsevier, 335-384.

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

[5] Fodor, J.A., Bever, T.G., & Garret, M.F. (1974), *The psychology of language*. New York: McGraw-Hill.

[6] Green, T. R. G. & Borning, A. (1990), The Generalized Unification Parser: Modelling the Parsing of Notations. In D. Diaper et al. (Eds.). *Human-Computer Interaction-INTERACT '90* . Amsterdam: North-Holland, 951-957.

[7] Guilford, J.P., Fruchter, B. (1978), *Fundamental statistics in psychology and education*, London: McGraw-Hill.

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

[9] 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.

[10] McNamara, T.P., Miller, D.L. & Bransford, J.D. (1991), Mental Models and Reading Comprehension. In: R. Barr, M.L. Kamil, P.B. Mosenthal & P.D. Pearson (Eds), *Handbook of Reading Research*, Vol. II, New York: Longman

[11] Melton, A. (1992), Specifying Internal, External, and Predictive Software Metrics, In: T. Denvir, R. Herman & R.W. Whitty (Eds), *Formal Aspects of Measurement*, London: Springer, 194-208.

[12] Petersen, G.M. van (1992), *Validation of axiomatic structure metrics for comprehensibility of Miranda type expressions*, M.Sc. Thesis, Enschede: University Twente.

[13] Robson, D.J., Bennett, K.H., Cornelius, B.J. & Munro, M. (1991), Approaches to Program Comprehension, *J. Systems Software, 14*, 79-84.

[14] Shepperd, M. & Ince, D. (1991), *Algebraic validation of software metrics*, Lecture Notes in Computer Science 550, Berlin: Springer, 343-363.

[15] Tian, J. & Zelkowitz, M.V. (1992), A formal program complexity model and its application, *J. Systems Software, 17*, 253-266.

[16] Turner, D. (1986), An Overview of Miranda. *Sigplan Notices, 21 (12)*, 158-166.

[17] Watt, D.A. (1990), *Programming Language Concepts and Paradigms*, New York: Prentice Hall.

[18] Weyuker, E. J. (1988), Evaluating Software Complexity Measures. *IEEE Trans. Software Engineering, SE-14 (9)*, 1357-1365.

[19] Whitty, R.W. (1992), Multi-dimensional Software Metrics, In: T. Denvir, R. Herman & R.W. Whitty (Eds), *Formal Aspects of Measurement*, London: Springer, 116-141

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

[21] Zwiers, J. (1989), *Compositionality, Concurrency and Partial Correctness*, Lecture Notes in Computer Science 321, Berlin: Springer.