Axiomatic Testing of Structure Metrics

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

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

Abstract

In this paper, axiomatic testing of software metrics will be described. The testing is based on representation axioms from the measurement theory. In a case study, the axioms are given for the formal relational structure and the empirical relational structure. Two approaches of axiomatic testing are elaborated: deterministic and probabilistic testing.

1. Introduction

In this paper, axioms from representational measurement theory will be utilized to establish the theoretical and empirical order of software entities with respect to some attribute. A simplified model for software measurement will be used (see Figure 1).

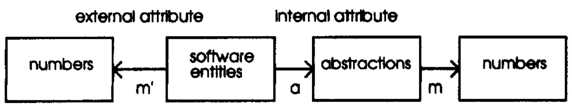

Figure 1. Model for software measurement

Software entities will be considered: products, processes or resources [4]. Data on an external attribute (e.g. maintainability, reusability) of these entities are collected with some measure m'. This external attribute will be related with some internal attributes, such as size or structure. The internal attribute is measured with a metric function m on abstractions of the software entities. The measure m is validated to the extent to which it preserves the order on the software entities obtained independently of m by the quantified criterion m' [15].

In a case study, axioms from the measurement theory will be tested, both formally and empirically. The case itself, comprehensibility and structure of type declarations, is of interest to researchers in the field of programming methodology. More general, the case is used to exemplify the application of representational measurement theory in software measurement and validation.

The representational approach has been used in software measurement [1, 3, 7, 15, 14, 20]. Some basic concepts in this measurement theory [9, 11, 12, 18] will be defined according to Roberts [17].

A *relational structure* is a (n+1)-tuple $(A, R_1,...,R_n)$, where A is a set, and $R_1,...,R_n$ are relations on A. A function f: $A \rightarrow A'$ is called a *homomorphism* from relational structure $(A, R_1,...,R_n)$ into relational structure $(A', R_1',...,R_n')$ if, for each $i \in 1..n$,

$$R_i(a_1,a_2,...,a_{r_i}) = R_i'(f(a_1), f(a_2), ..., f(a_{r_i})).$$

A homomorphism is an order preserving mapping. The triple $((A, R_1,...,R_n), (A', R_1',...,R_n'),f)$ is said to be a *scale*. If $(\mathcal{A},\mathcal{B},f)$ is a scale and φ is a function such that $(\mathcal{A},\mathcal{B},\varphi.f)$ is a scale as well, then φ is said to be an *admissible transformation of scale*. The *representation* $\mathcal{A} \rightarrow \mathcal{B}$ is *regular* if all scales $(\mathcal{A},\mathcal{B},f)$ are related via an admissible transformation of scale. The class of admissible transformations of scale of a regular representation defines the *scale type* of the representation. Some common scale types are: absolute, ratio, interval, ordinal and nominal scale [17, p. 64]. The focus in the case study is on *ordinal* scales, which are defined by monotone increasing transformation functions.

The aim of software measurement is to enable the comparison of software entities with respect to some attribute. As given in the model of figure 1, there are four relational structures. The outmost structures are numerical relational structures. The software entities with their relations form the empirical relational structure. The fourth relational structure involves the abstractions. Axioms, that will be tested, state sufficient

0-8186-5865-7/94 $4.00 © 1994 IEEE

conditions for the existence of a regular scale. By investigating the axioms, the representation and measurement scale of these structures can be established.

In order to make the discussion of axiomatic testing concrete, a case study will be presented related to a specific kind of software documentation: type declarations. The programmer provides explicit information about the type of the objects in the program. This form of documentation not only may have an impact on the reliability of the software, but also on the comprehensibility to human readers of the programs (reviewers, maintenance programmers). The software entities are type declarations in the functional programming language Miranda[1] [19]. Type declarations themselves have a certain degree of (cognitive) complexity: they are easy or difficult to comprehend. 'The comprehensibility' will be taken as the external attribute. The internal attribute is 'the structure' of type expressions. The relationship between the comprehensibility of type expressions and their structural properties will be investigated: first, by establishing the scale of measurement of the internal attribute and the external attribute, and then by investigating the correspondence between both measurements.

This paper is organised as follows. In section 2, more details about the software entities in the case study, the type declarations, will be given. In the subsequent section, the modelling of the structure of type expressions is elaborated. The actual collection of data on the external attribute, the comprehensibility, will be described in section 4, with an exemplification of the deterministic and probabilistic testing of axioms. The final section discusses some results obtained with this approach.

2. The case study

The software entities considered in the case study, type expressions, will be introduced. In an imperative programming language like Modula-2 or Pascal, the heading of a procedure declares the type. For example, the heading of the function procedure *IsDigit* is:

```
PROCEDURE IsDigit (C: CHAR): BOOLEAN;
```

In the case study, type expressions in the functional programming language Miranda are considered. The type declaration of the function *isdigit* is (denoted with :: and on the right hand side a type expression):

```
isdigit :: char → bool
```

A function type is denoted with the symbol "→". The function *split* has a more complex type: *split* returns a tuple with two lists of numbers, one with elements of a given list that satisfy a predicate, and the second list with elements that do not.

```
split :: (num → bool) → [num] → ([num],[num])
```

The type of the predicate, the first argument of the function *split*, is a function type (num → bool). This argument is enclosed by round grouping brackets. The type of the second argument [num] is a list type, a list of elements of type num. A list type is denoted with square brackets. The type of the result of the function is a tuple type with two components, each of type [num]. A tuple type is denoted with round brackets and component types separated by a comma. It is possible to define type *synonyms*, e.g. numlist == [num]. The type of function *split* with this synonym is:

```
split :: (num→bool)→numlist →(numlist,numlist)
```

The canonical form of the two given types of the function *split*, as used by the type checker, is the same (and equal to the first one).

Type declarations form an important clue to the understanding of functions in a program. They give a partial specification of the function: the type of its arguments and the type of the result. The complexity of the type declaration might give an indication of the complexity of the task to be performed by the function (e.g.[5]).

In the 'real world model' [13], restrictions will be imposed on the 'real world' entities and phenomena. In the case study the type expressions will be restricted to three standard types: *char*, *num* and *bool*, and three type constructors: the function type, the tuple type, and the list type (for homogeneous lists). Furthermore, neither type variables nor abstract data types are considered. Also, the influence of the naming of types and typographic issues will not be considered. Type synonyms are not considered. In other words, these aspects will be kept invariant in the case study. Type expressions with these restrictions will be called *simple* type expressions.

Type expressions are studied in the context of programs developed in an academic environment. It is evident that comprehensibility depends on the experience of the reader. The case study is carried out with novice Miranda programmers with corresponding proficiency. Only structural properties of simple type expressions in

[1] Miranda is a trademark of Research Software Ltd.

Miranda in relation with their comprehensibility to novice programmers are examined.

3. The theoretical order

In this section, the modelling of type expressions is described. The abstraction of type expressions is defined, a relation on abstract type expressions and a structure metric (cf. [15]). Structure metrics are based on the compositionality of the structural [6]. On the basis of these definitions, the scale of measurement is established.

3.1 The abstraction

A relational structure $(A, R_1,...,R_n)$ is defined. Set A consists of abstract type expressions; $R_1,...,R_n$ are relations on abstract type expressions. In some cases, the corresponding operation of a relation will be used in the relational structure (cf. [17: p. 41]). These operations are called *concatenation operators* or *constructors.*

First, the mapping of simple type expressions to abstract type expressions is defined. This *abstraction* implies the following rules:
1. the order between components in a tuple type expression is disregarded
2. grouping brackets around a tuple type expression with one component are disregarded
3. grouping brackets implied by the right associativity of the function type constructor are disregarded.

For abstract type expressions the following data structure will be used as model:

```
texp::= L texp | F [texp] | T {texp} | C | N | B
```

with respectively: L the list type constructor; F the function type constructor; T the tuple type constructor; C the standard type *char*; N for *num* and B for *bool*. Next to the constructors, [texp] denotes an ordered list of abstract type expressions, and {texp} denotes a multiset. Some examples of the abstraction are given in Table 1.

	type expression	abstraction
t_a	((num → [bool]), bool)	T { F [N, L B], B }
t_b	(bool, num → [bool])	T { B, F [N, L B]}
t_c	num → ([bool] → bool)	F [N, L B, B]
t_d	num → [bool] → bool	F [N, L B, B]
t_e	(num → [bool]) → bool	F [F [N, L B], B]
t_f	(num→bool)→[num]→([num],[num])	F [F [N,B], L N, T {L N, L N}]

Table 1. Example type expressions with abstractions

The abstractions of t_a and t_b are the same: the round brackets in (num → [bool]) are disregarded (rule 2), as the order of components in the tuple (rule 1). The abstractions of t_c and t_d are the same. The abstraction of t_c is not F [N, F [L B, B]], since the round brackets in ([bool] → bool) are disregarded (rule 3). In other words, the type of the result of a function is not allowed to be a function type. The abstractions of t_d and t_e are different, since the grouping brackets in t_e can not be disregarded (→ is right associative). With t_f, the abstraction of the type of the example function *split* is given.

There are alternative abstractions; for example, to restrict the function type to F $[t_1, t_2]$ (only two types in a function type); or to disregard the order of the arguments of functions. These alternatives are discussed in [2]. The choice between abstractions of entities is determined by the actual use of the abstractions: the establishment of a good correspondence between an internal attribute based on these abstractions, and an external attribute of the entities.

3.2 The containment relation

The containment on abstract type expressions, denoted by $\prec$, will be defined. Let a and b be abstract type expressions, with concatenation operators $\oplus$ and $\otimes$ respectively, and with maximal subexpressions $a_1,...,a_n$ and $b_1,...b_m$ respectively, as depicted in Figure 2.

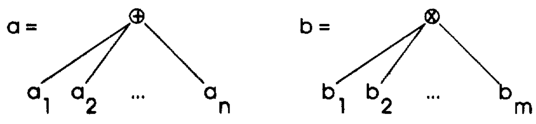

Figure 2. Two abstract type expressions

Then a $\prec$ b iff a is contained in b in the following sense: a is contained in b if a is contained in b_i for some i. Moreover, if $\oplus = \otimes$ then a is contained in b if each a_i is contained in some $b_{i'}$, subject to the conditions that 1',...,n' are pairwise different, and if $\oplus$ = F then [1',...,n'] is ordered. The containment relation is a partial order.

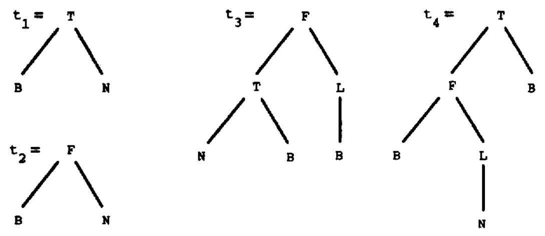

Figure 3. Example abstract type expressions

Consider, for example, the set of abstract type expressions in Figure 3. The containment relation of these type expressions, $(\{t_1,t_2,t_3,t_4\}, \{(t_1,t_3), (t_1,t_4), (t_2,t_4)\})$, is given in the partial order diagram (Hasse diagram) in Figure 4.

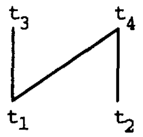

Figure 4. Partial order with $\prec$ on example type expressions of Figure 3

With the definition of this containment relation, the formal relational structure (texp, $\prec$, L, F, T, C, N, B) for abstract type expressions has been described.

3.3 Extension of the containment relation and ordinal scale

For the relational structure from the previous section, a measurement scale will be considered, based on theorems from measurement theory [17]. Only ordinal measurement will be discussed here. The following theorem will be used:

Suppose A is a countable set and R is a binary relation on A. If f is a real-valued function on A which satisfies

$$a \mathrel{R} b \Leftrightarrow f(a) \leq f(b) \qquad (.\ 1)$$

then ((A, R), (Re, $\leq$), f) is an ordinal scale [17, p. 110].

For abstract type expressions, a linear *structure metric* function m is defined, with all constants $c_i \geq 0$:

$$m(C) = c_C \qquad (.\ 2)$$
$$m(N) = c_N \qquad (.\ 3)$$
$$m(B) = c_B \qquad (.\ 4)$$
$$m(T\{t_1,...,t_n\}) = c_T + m(t_1) + ... + m(t_n) \qquad (.\ 5)$$
$$m(L\ t) = c_L + m(t) \qquad (.\ 6)$$
$$m(F[t_1,...,t_n]) = c_F + m(t_1) + ... + m(t_n) \qquad (.\ 7)$$

The theorem above is not applicable for this function m and the containment relation $\prec$ on type expressions, because equation 1 is not satisfied ($\prec$ is a partial order). Therefore, with this function m a new relation $\prec_m$ on type expressions is defined as follows:

$$t_a \prec_m t_b \Leftrightarrow m(t_a) \leq m(t_b) \qquad (.\ 8)$$

The relation $\prec_m$ is an *extension* of the containment relation $\prec$, i.e.

$$t_a \prec t_b \Rightarrow t_a \prec_m t_b \qquad (.\ 9)$$

From the theorem above it follows that ((texp,$\prec_m$), (Re, $\leq$), m) is an ordinal scale.

In this section, an abstraction of type expressions and a containment relation on abstract type expressions have been defined. An extension of this relation derived from a structure metric function provides measurement of the internal attribute structure of type expressions on an ordinal scale. This allows the investigation of a correspondence of the extension with the empirical order as given by the quantified criterion, which also maps on (Re,$\leq$) (see subsequent section). This approach differs from the one proposed by Fenton [8]. For a partial order on flowgraphs, Fenton defines a mapping of flowgraphs to (N,|), where N is the set of natural numbers and | is the relation 'divides without remainder', instead of a mapping to (Re,$\leq$), in order to satisfy the representation condition (equation 1). In the following section, the empirical order of type expressions will be discussed.

4. The empirical order

In this section, the order of type expressions will be established with respect to the external attribute comprehensibility. The conditions for an ordinal scale are investigated.

There are several approaches to the measurement of comprehensibility of programs. In the case study, one measure has been chosen for the comprehensibility of type expressions [2]: the time in seconds needed for a subject to read a given type expression and to conceive and typewrite a (function) definition with exactly this type in the 'standard' programming environment. The time between showing the type expression on the screen and the completion of the answer is measured automatically. Afterwards, with the type checker of the programming system, the answer is marked as correct or incorrect. This time measurement will be used as criterion for the comprehensibility The data have been collected in controlled experiments. The subjects in the experiment are novice programmers, all first year students in computer science. Two data sets are used, each based on responses of 14 subjects to 42 type expressions (per data set 588 responses). Dataset1 consists of responses of 14 subjects to 16 type expressions, with a total of 241 correct responses; dataset2 is based on responses of 14 other subjects to another set of 16 type expressions, with a total of 347 correct responses. The type expressions are offered to the subjects in random order. Of the 42 questions in the original experiment, expressions with type variables have not been considered here, neither have questions with less than 6 correct answers. In table 2 the type expressions of dataset1 are given. Further details of the experiments can be found in [2, 16].

rank	nr	type expression	# correct	average (sec)	std dev (sec)
16	20	bool→char→bool	6	50.0	27.6
15	27	(num,bool) → (num,bool)	10	44.7	24.7
14	12	num→char→char	7	35.0	19.6
13	13	[(num,bool)]	12	30.0	9.8
12	18	bool → num	14	28.9	12.8
11	5	[char]	12	24.8	6.2
10	34	(num,bool,char)	13	24.7	7.2
9	15	num → bool	12	23.3	9.5
8	28	char → char	7	23.1	6.7
7	17	char → bool	8	21.8	8.3
6	26	num → num	10	19.7	10.6
5	14	(num,bool)	14	18.8	5.4
4	3	bool	14	17.7	9.6
3	2	num	14	14.6	5.5
2	41	(num,num)	13	13.9	3.1
1	1	char	13	12.8	3.2

Table 2. Ranking of type expressions in dataset1 according to the average time

The following approaches in the analysis of the data will be used. Firstly, a global analysis will be given based on the average time measured for each type expression. Secondly, an axiomatic analysis of the relative preference of each subject between pairs of type expressions will be described. Finally, an axiomatic analysis based on the relative frequencies of these preferences will be considered.

4.1 Global analysis of the empirical order

For each type expression, the average time for all correct responses has been calculated. The data for the first set are given in Table 2. In Figure 5, for a subset of type expressions of dataset1, the empirical order based on the average time and the theoretical partial order are compared. The empirical order for this subset, except the value obtained on type expression 41, is an extension of the partial order of the 9 abstract type expressions. Taking into account a rather large standard deviation in the measured values, there is reasonable agreement between the theoretical order and the empirical order based on the average time. However, the scale type of the empirical order itself is not yet known from this analysis. For this purpose, the properties of this order have to be analysed by examining the axioms, as has been done for the theoretical order in the previous section.

4.2 Axiomatic analysis of the empirical order

Two types of axiomatic analysis will be ensued: a deterministic analysis and a probabilistic analysis. Each of them aims at establishing the representation of the empirical order by testing the axioms from the theorems. The theory of the deterministic analysis can be found in Krantz *et al.* [11]; of the probabilistic analysis in Suppes *et al.* [18]. In this section, Roberts [17] will be used as the main reference. It should be expected that the comprehensibility measure in the experiment is on an ordinal scale. In that case the data should be conform a (strict) weak order.

4.2.1 Deterministic axiomatic analysis

On the basis of the time measurement (in seconds) for each type expression per subject, it is possible to define the relation R for all type expressions a, b in the data set A: aRb $\Leftrightarrow$ $t_a > t_b$. This relational structure (A,R) represents the 'preference' of each subject in the indication of the most difficult type expression.

The preference structure (A,R) can be represented in the *preference matrix* (A,p) defined as, $\forall$ a,b $\in$ A:

$$p_{ab} = 1 \Leftrightarrow aRb \quad \text{and} \quad p_{ab} = 0 \Leftrightarrow \neg\, aRb \qquad (.\ 10)$$

The ranking of correctly answered questions per subject is determined. All these individual rankings form a profile, i.e. a list of k rankings (k is the number of subjects). In the experiment, not all individual rankings are complete, since not all questions have been answered correctly. There are only 2 subjects for each data set with a complete ranking. A reduction of dataset1 to a subset of 7 questions (subset1 = {12, 13, 15, 18, 20, 27, 34}) results in 5 complete rankings; also, a reduction of dataset2 to 7 questions (subset2 = {115, 118, 123, 124, 127, 129, 132}) results in 5 complete rankings.

A *group preference structure* (A,M) from a list of complete individual preference structures can be derived, for example according to the *simple majority rule*, defined as follows [17, p. 118]:

$$aMb \Leftrightarrow \#\, aRb > (\#\, aRb + \#\, bRa)/2 \qquad (.\ 11)$$

where # xRy is the number of relations R which contain (x,y).

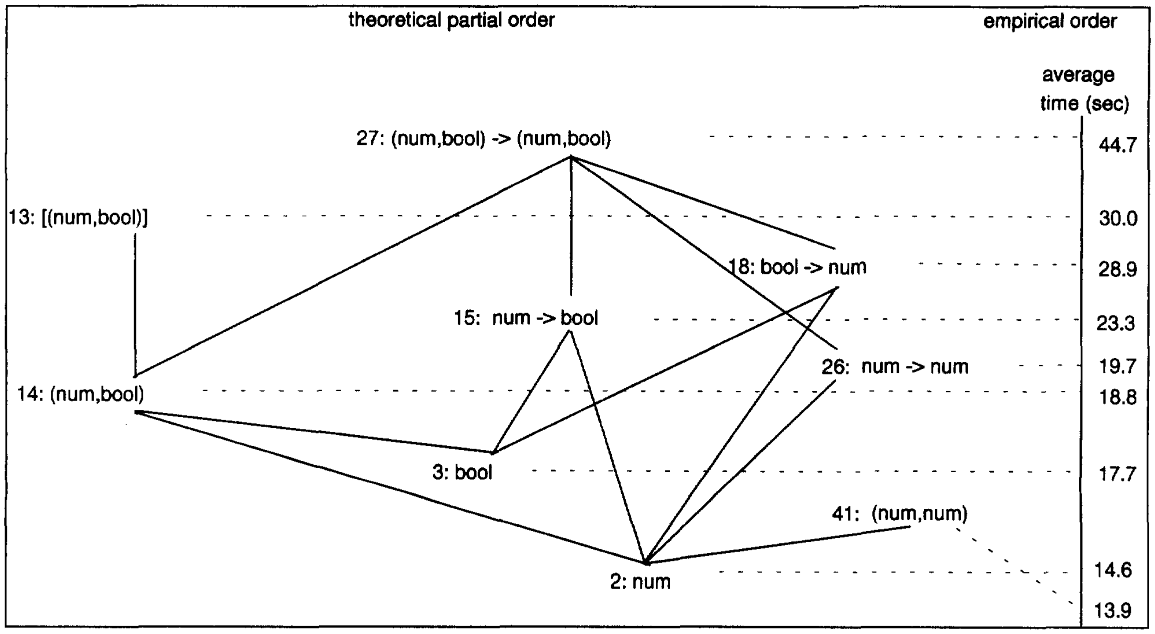

Figure 5. Theoretical partial order in Hasse diagram and empirical order of subset of type expressions in dataset1

The group preference matrix of subset1 based on the simple majority rule is given in Table 3. In total 35 correct responses have been used.

A group ranking can be obtained from the group preference structure if the data are consistent: there are no intransitivity's (i.e. a preference cycle: $p_{ab} = 1 \wedge p_{bc} = 1 \wedge p_{ca} = 1$) allowed.

nr	12	13	15	18	20	27	34
12	0	1	1	1	0	0	1
13	0	0	1	1	0	1	1
15	0	0	0	0	0	0	0
18	0	0	1	0	0	0	1
20	1	1	1	1	0	1	1
27	1	0	1	1	0	0	1
34	0	0	1	0	0	0	0

Table 3. Group preference (A,M) for 7 type expressions of dataset1 (k=5)

For this subset there are inconsistencies in the group preference structure. The three type expressions that are not transitive are: 12: (num $\rightarrow$ char $\rightarrow$ char); 13: [(num,bool)]; and 27: (num,bool) $\rightarrow$ (num,bool).

The group preference structure of the second subset is consistent. It is a strict weak order (asymmetric and negatively transitive). An ordinal function m for this subset is defined as follows [17, p. 105]:

$$m(x) = \#\{ y \in A \text{ such that } x\, R\, y \} \qquad (.\ 12)$$

The function m for the subset of type expressions of dataset2 is given in Table 4.

nr	function		
123	m ([char] $\rightarrow$ bool)	=	6
124	m (bool $\rightarrow$ [char])	=	5
129	m ([char])	=	4
132	m (bool $\rightarrow$ char)	=	3
127	m (char $\rightarrow$ bool $\rightarrow$ bool)	=	2
115	m (bool)	=	1
118	m (char)	=	0

Table 4. Function m for type expressions in subset2

For the first subset, this function yields the same value for each of the type expressions 12, 13 and 27, resulting in a violation of the representation condition (equation 1).

4.2.2 Probabilistic axiomatic analysis

A major disadvantage of the analysis in the previous section is that only complete preference structures can be taken into account. With a probabilistic analysis this can be circumvented. It is possible to calculate the *probability matrix* [17, p. 273] with relative frequencies based on all correctly answered questions:

$p_{ab} = (\# aRb) / (\# aRb + \# \neg(aRb))$, if $a \neq b$ (. 13)
$p_{ab} = 0.5$, if $a = b$ (. 14)

From this it can be seen that: $\forall a,b \in A$: $p_{ab} + p_{ba} = 1$. Such a probability matrix represents a *forced choice pair comparison structure* (A,p). This structure (A,p) is *weak stochastic transitive* if, $\forall a,b,c \in A$:

$p_{ab} \geq 0.5 \wedge p_{bc} \geq 0.5 \Rightarrow p_{ac} \geq 0.5$ (. 15)

A weak order (A,W), associated with a weak stochastic transitive structure (A,p), is given by W defined on A by

$aWb \Leftrightarrow p_{ab} \geq p_{ba}$ (. 16)

As an example, in Table 5 the probability matrix is given for the same subset of type expressions as in the previous section. The matrix can be compared with the group preference matrix of table 3. However, the matrix presented here has been calculated with data of all 14 subjects. In total 74 correct responses have been used. This probability structure is weak stochastic transitive and hence consistent, contrary to the group preference of 5 subjects.

nr	12	13	15	18	20	27	34
12	.50	.67	1.0	.71	.40	.43	.71
13	.33	.50	.60	.58	.33	.44	.64
15	.0	.40	.50	.25	.0	.0	.42
18	.29	.42	.75	.50	.33	.30	.54
20	.60	.67	1.0	.67	.50	.60	.83
27	.57	.56	1.0	.70	.40	.50	.89
34	.29	.36	.58	.46	.17	.11	.50

Table 5. Probability matrix for 7 type expressions of dataset1 (k=14)

On the basis of this probability structure for these type expressions, an associated weak order can be calculated with a ranking (see Table 6).

rank	type expressions
7	20: bool → char → bool
6	27: (num,bool)→(num,bool)
5	12: num → char → char
4	13: [(num,bool)]
3	18: bool → num
2	34: (num,bool,char)
1	15: num → bool

Table 6. Ranking of 7 type expressions based on associated weak order (k=14)

In the previous analysis, no attention has been given to *measurement errors* and the significance of the experimental data. For the probability matrix from this data set (Table 5), the significance of the relative frequencies has been calculated. The sign test has been used[2] [10]. A significance of $\alpha < .09$ will be achieved if 10 out of 14 subjects show the same sign of the difference between the time measured for two type expressions t_a and t_b, which presumes a probability $p_{ab} \geq 0.71$. For the probability of the type expressions in subset1, the structure (A,W) is calculated with

$aWb \Leftrightarrow p_{ab} \geq \lambda$ (. 17)

with threshold probability $\lambda = 0.75$. The structure obtained in this case is not a weak order, however it satisfies the axioms for a *semiorder*, which are the following [17, p. 250]:

$\neg\, aRa$ (. 18)
$aRb \wedge cRd \Rightarrow (aRd \vee cRb)$ (. 19)
$aRb \wedge bRc \Rightarrow (aRd \vee dRc)$ (. 20)

A weak order (A,W) associated with the semiorder (A,R) can be obtained with W defined on A by [17, p. 256]:

$aWb \Leftrightarrow \forall c \in A: (bRc \Rightarrow aRc) \wedge (cRa \Rightarrow cRb)$ (. 21)

For the semiorder obtained above, the associated weak order has been calculated. A ranking for this weak order is given in Table 7, with ties at ranks 4-5 and 6-7 (resulting respectively in rank 4.5 and 6.5).

[2] The Wilcoxon signed ranks test is not applicable because the rankings are not complete for all subjects.

rank	type expressions
6.5	20: bool → char → bool 27: (num, bool) → (num, bool)
4.5	12: num → char → char 18: bool → num
3	13: [(num, bool)]
2	34: (num, bool, char)
1	15: num → bool

Table 7. Ranking of a subset of 7 type expressions based on the associated weak order of the semiorder (λ=0.75, k=14)

From the previous analysis of the empirical order of type expressions with respect to the external attribute comprehensibility, it can be concluded that, for subsets of type expressions, the measurement of time to find an instance of a given type, results in an ordinal scale.

5. Discussion

It has been shown that type expressions can be measured on an ordinal scale with respect to the internal attribute structure by defining an extension of a containment relation on abstract type expressions.

In the case study, the comprehensibility of simple type expressions has been operationalized as a time measurement. The ranking of the average time is in reasonable agreement with a weak order extension of the partial order obtained for the corresponding abstract type expressions. Axiomatic analysis has been used to localise inconsistencies in the experimental data: an example has been given of an intransitive group preference. An ordinal measure has been calculated for a consistent data set. Incomplete data sets have been analysed with a probabilistic consistency axiom: the weak stochastic transitivity. An ordinal measure has been established based on these probabilistic data. Measurement errors have been treated with a threshold probability and semiorders. The order obtained in this way shows a deviation of the previous order and appears to have more ties.

Subsequently, the correspondence between the two measurements can be established now. There are two steps which have been described in a previous study [2]. Firstly, the structure metric function m defined in section 3.3 is calibrated, resulting in values for c_i. This can be done with standard linear regression techniques. Secondly, this calibrated function is used in the prediction of the comprehensibility values. The forecasting efficiency of the prediction has been established.

Another important aspect is the use of the approach outlined in this paper to other software entities with other attributes. There seems to be at least one important field where this approach could be successful. This is the domain of complexity measures based on flowgraph modelling. An ordering of flowgraphs is given by Bache (see [7]). A containment based order has been defined by Melton [8, 15], and a formal axiomatic validation is presented by Zuse [20]. An experimental axiomatic testing could be carried out along the framework described in this paper, e.g. for maintainability and structural properties.

The main point presented in this paper is the role of representation axioms in the diagnostic testing [12] of the order of attributes of software entities. Inconsistencies can be localised. They may hint at anomalies in the experiments or weaknesses in the theory: they can be used in the development of the conceptual domain, e.g. in the choice of alternative abstractions. It has been shown that axiomatic testing may well contribute to the validation of software metrics, both formally and empirically.

References

1. Baker, A.L., Bieman, J.M., Fenton, N., Gustafson, D.A., Melton, A. & Whitty, R. (1990), A Philosophy for Software Measurement, *J. Systems Software*, 12, 277-281.
2. Berg, K.G. van den, Broek, P.M. van den & Petersen, G.M. van (1993), Validation of Structure Metrics: A Case Study. *Proceedings of International Software Metrics Symposium METRICS 93*, Washington: IEEE Computer Society Press, 92-99.
3. Bieman, J., Fenton, N.E., Gustafson, D., Melton, A. & Whitty, R. (1992), Moving from Philosophy to Practice in Software Measurement. In: T. Denvir, R. Herman & R.W. Whitty (Eds), *Formal Aspects of Measurement*, London: Springer, 38-59.
4. Bush, M.E. & Fenton, N.E. (1990), Software Measurement: A Conceptual Framework, *J. Systems Software*, 12, 223-231.
5. Cardelli, L. & Wegner, P. (1985). On Understanding Types, Data Abstraction, and Polymorphism. *ACM Computing Surveys*, 17 (4) 471-522.
6. 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.
7. Fenton, N.E. (1991), *Software Metrics: A Rigorous Approach*, London: Chapman & Hall.

8. Fenton, N.E. (1992). When a software measure is not a measure. *Software Engineering Journal, Sept, 357-362.*
9. Finkelstein, L., & Leaning, M.S. (1984). A Review of the Fundamental Concepts of Measurement. *Measurement,* 2(1), 25-34.
10. Guilford, J.P. & Fruchter, B. (1978), *Fundamental statistics in psychology and education,* London: McGraw-Hill .
11. Krantz, D.H., Luce, R.D., Suppes, P., & Tversky, A. (1971). *Foundations of Measurement,* Volume I. New York: Academic Press.
12. Luce, R.D., Krantz, D.H., Suppes, P., & Tversky, A. (1990). *Foundations of Measurement,* Volume III. San Diego: Academic Press.
13. Maki, D.P & Thompson M. (1973). *Mathematical Models and Applications.* Englewood Cliffs, Prentice-Hall.
14. 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.
15. Melton, A.C., Gustafson, D.A., Bieman, J.M. & Baker, A.L. (1990), A Mathematical Perspective for Software Measures Research, *Software Engineering Journal,* Sept, 246-254.
16. Petersen, G.M. van (1992), *Validation of Axiomatic Structure Metrics for the Comprehensibility of Miranda Type Expressions.* MSc thesis, University Twente, Enschede.
17. Roberts, F.S. (1979), *Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences.* Encyclopaedia of Mathematics and Its Applications, Volume 7, London: Addison-Wesley.
18. Suppes, P., Krantz, D.H., Luce, R.D., & Tversky, A. (1989). *Foundations of Measurement,* Volume II. New York: Academic Press.
19. Turner, D. (1986). An Overview of Miranda. *Sigplan Notices, 21 (12),* 158-166
20. Zuse, H. (1992). Properties of Software Measures. *Software Quality Journal,* 1, 225-260.