Chapter 8

8. Validation in the Software Metric Development Process [1]

In this chapter the validation of software metrics will be examined. Two approaches will be combined: representational measurement theory and a validation network scheme. The development process of a software metric will be described, together with validities for the three phases of the metric development process. Representation axioms from measurement theory are used both for the formal and empirical validation. The differentiation of validities according to these phases unifies several validation approaches found in the software metric's literature.

8.1 Introduction

As can be concluded from the plethora of software metrics, it is rather easy to conceive some software metric and to obtain numbers with such a metric, for example in the field of complexity measures. However, it is less clear that all these metrics are really good measures. To establish the quality of measures they have to be validated. It has been remarked that there are as many metrics as there are computer scientists[2]. A paraphrase of this statement is that there are as many types of validation as there are software metrics. Validation is defined as assessing the extent to which a measure really measures what it purports to measure (Fenton, 1991). However, this is a rather tautological formulation (Berka, 1983), and validation has to be operationalized in practice.

[1] This chapter is a shortened version of: K.G.van den Berg & P.M. van den Broek (1995), Axiomatic Validation in the Software Metric Development Process, in: A.Melton (Ed.), Software Measurement: Understanding Software Engineering, London: Thomson, Chapter 10.

[2] Ascribed to S.D.Conte

In this chapter, the development of a software metric will be traced with an explication of different aspects of validation. A simplified framework for software measurement will be used (see Figure 8.1).

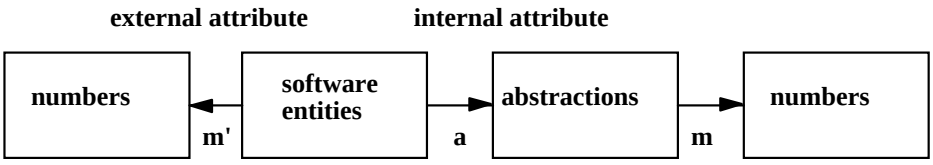

Figure 8.1 Framework for software measurement

Software entities will be considered: products, processes or resources (Bush & Fenton, 1990). Data on an external attribute (e.g., maintainability, reusability) of these entities are collected with some measure *m'*, the quantified criterion (Melton *et al.*, 1990). 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.

Kaposi (1990) has given an account of the role of measurement theory in software engineering. Five parts in the planning of measurement are distinguished: 1. The problem definition: designating the target objects and key properties that must be measured. 2. The modelling: a model description of the target set with reference to the key properties. 3. The forming of the empirical relational system: describing the model by means of an observable relation between the objects in terms of the selected key properties. 4. The definition of the formal relational system: selecting the system in which the measured results are to be represented. 5. The validation of the results of the measurements.

In this chapter, two approaches will be brought together: the validity network scheme, which resembles Kaposi's analysis, and the representational measurement theory. In the validity network scheme, aspects of validity are differentiated for subsequent phases of the research process. In a case study, axioms from the measurement theory will be validated, both formally and empirically, according to this scheme. The case itself is of interest to researchers in the field of programming methodology (van den Berg *et al.*, 1993). More general, the case is used to exemplify the application of representational measurement theory and aspects of validation in software measurement.

As remainder of this section, the representational measurement will be introduced briefly (section 8.1.1), followed by the validity network scheme (section 8.1.2) and the case study (section 8.1.3).

8.1.1 The representational measurement theory

The representational approach has been used in software measurement (e.g., Baker *et al.*, 1990; Fenton, 1991; Melton, 1990; Bieman *et al.*, 1992; Melton, 1992; Zuse, 1992). Some basic concepts in this measurement theory (Krantz *et al.*, 1971; Finkelstein & Leaning, 1984; Suppes *et al.*, 1989; Luce *et al.*, 1990) will be defined according to Roberts (1979). A pivotal concept is the order preserving mapping between relational structures. A short introduction to the representational measurement theory has been given in Chapter 7.1.

8.1.2 The validity network scheme

According to Brinberg and McGrath (1985), three *domains* can be distinguished in the research process: the *substantive* domain, the *conceptual* domain and the *methodological* domain. Each domain is defined by its elements, the relations between the elements, and the embedding system. The embedding system refers to the set of assumptions within which these elements and relations are studied.

In software measurement, the substantive domain consists of the empirical relational structure in the framework (Figure 8.1), together with the embedding system: the actual *context* of the software entities (e.g., industry, training). The conceptual domain consists of the other relational structures in the framework. The embedding system in the conceptual domain is called the *paradigm*. For example, structure metrics are based on the assumption of the compositionality of the structural properties. The methodological domain is primarily concerned with the mapping of the empirical relational structure into a numerical relational structure. The embedding system in this domain is the research *strategy*, for example the use of field studies or controlled experiments.

The research process itself consists of three *phases*: the *generative* or prestudy phase, the *executive* or central phase, and the *interpretative* or generalisation phase (Table 8.1).

domain / phase	substantive	conceptual	methodological
generative	valuation validities		
executive	correspondence validities		
interpretative	generalisation validities		

Table 8.1 The validity network scheme

In each domain of each phase, there are specific aspects of validity depicted in the validity network scheme. In the generative phase, the *valuation* validities are of primary concern, in the executive phase the *correspondence* validities, and in the interpretative phase the *generalisation* validities. A description of the various aspects of these validities will be presented in the elaboration of the case study.

8.1.3 The case study [3]

In order to make the discussion of the different aspects of validation concrete, a case study will be presented related to a specific kind of software entities: software documentation. Proper documentation presumably has an impact on important quality aspects, such as maintainability and reusability. There is an interest in objective data on the impact of documentation. For the program code, the documentation problem is obvious. Besides documentation in natural language, there is a tendency to formalise documentation. On procedural level this can be done with for example preconditions and postconditions. Another possibility is the use of explicit typing by the programmer. In this case, the programmer provides information about the type of the objects in the program. (This is opposed to implicit typing, where the computer carries out the check of types as can be derived from the code.) 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). In the case study, documentation in the form of explicit typing will be considered. The software entities are type expressions in the functional programming language Miranda[4]. Type expressions 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.

8.1.4 Overview

This chapter is organised as follows. First, the generative phase of a software metric will be described (section 8.2). More details about the software entities in the case study, the type expressions, will be given. Furthermore, the modelling of the structure of type expressions and the measurement of the compre-

[3] This is the same case study as described in Chapter 7 of this thesis

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

hensibility will be elaborated on. The subsequent section 8.3 will deal with the executive phase in the development of the software metric. The actual collection of data on the external attribute, the comprehensibility, will be described. The deterministic and probabilistic testing of axioms will be exemplified. The structure metric function will be calibrated and used for prediction of the comprehensibility. The interpretative phase in the following section 8.4 will elaborate on the generalisation of the results obtained in the foregoing phases. The final section 8.5 discusses the relation of axiomatic validation presented in this chapter to other validation approaches.

8.2 The generative phase

Consecutively, the substantive domain, the conceptual domain and the methodological domain in the generative phase will be described. For each domain, the elements, the relations and the embedding system will be given. This section will be concluded with a discussion of the valuation validities in this phase.

8.2.1 The substantive domain

An outline of the substantive domain implies the *phenomena*, the *observed patterns* and the *context* in 'the real world'. The phenomenon to be studied is the comprehensibility of type expressions, which has been introduced in Chapter 7.2. For example, the type of a function *split* is:

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

Several observations have been made with respect to the role of explicit types in programming, e.g.

> *Miranda scripts often contain type declarations as these are useful for documentation and provide an extra check, since the type checker will complain if the declared type is inconsistent with the inferred one. (Turner, 1986)*

> *Types impose constraints that help to enforce correctness. Typing enforces a programming discipline on the programmer that makes programs more structured and easier to read. (Cardelli & Wegner, 1985)*

> *Judicious placement of type signatures is a good idea, since it improves readability and helps bring programming errors to light. (Hudak & Fasel, 1992)*

Typing would make the programmer think about what kind of parameters a function will be used for and, also, would provide more information about how the program worked to anyone reading or maintaining it at a later stage. (Kosky, 1988)

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.

In the 'real world model' (Maki & Thompson, 1973) restrictions will be imposed on the 'real world' entities and phenomena. In the case study the type expressions will be restricted to so called *simple* type expressions (no type variables, no type synonyms, no abstract data types: see Chapter 7.2).

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

8.2.2 The conceptual domain

The second domain, the conceptual domain, implies the *concepts*, the *relations*, and the *conceptual paradigm*. The concepts will be given in a relational structure with relations on abstract type expressions. The conceptual paradigm is the representational measurement theory as described above, and the compositionality of the structural properties, as expressed in structure metrics (Fenton & Kaposi, 1989).

8.2.2.1 The abstraction

In the conceptual domain 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 Roberts, 1979: 41). These operations are called *concatenation operators* or *constructors*.

The mapping of simple type expressions to abstract type expressions is described in Chapter 7.3.1. For example, the abstraction of the type of the function *split* is:

```
F [ F [N,B], L N , T {L N, L N}]
```

with respectively: L the list type constructor; F the function type constructor; T the tuple type constructor; C the standard type *char* (not used is this example); N for *num* and B for *bool*. Next to the constructors, [...] denotes an ordered list of abstract type expressions, and {...} denotes a multiset.

There are alternative abstractions discussed in van den Berg *et al.* (1993). 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.

8.2.2.2 The containment relation and the metric function

The containment relation on abstract type expressions, denoted by $\prec$, has been defined in Chapter 7.3.2. The containment relation $\prec$ on type expressions is a partial order.

For abstract type expressions, a linear *structure metric* function *m* is defined in Chapter 7.3.3 :

$$
\begin{array}{lcl}
m(C) & = & c_C \\
m(N) & = & c_N \\
m(B) & = & c_B \\
m(T\{t_1,\ldots,t_n\}) & = & c_T + m(t_1) + \ldots + m(t_n) \\
m(L\ t) & = & c_L + m(t) \\
m(F[t_1,\ldots,t_n]) & = & c_F + m(t_1) + \ldots + m(t_n)
\end{array}
$$

With this function *m*, a new relation $\prec_m$ on type expressions is defined as follows:

$$t_a \prec_m t_b \quad \Leftrightarrow \quad m(t_a) \leq m(t_b)$$

The relation $\prec_m$ is an *extension* of the containment relation. From a measurement theorem it has been shown that ((*texp* ,$\prec_m$), (*Re,* $\leq$), *m*) is an ordinal scale.

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,*$\leq$). In the following section, the measurement of the external attribute, i.e. the comprehensibility of type expressions, will be discussed.

8.2.3 The methodological domain

In this section, the methodological domain - which comprises the *measures*, *comparison techniques* and the *research strategy* - is described. The collection of data on the external attribute of the software entities will be addressed. The external attribute has to be operationalized by some measure.

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 (van den Berg *et al.*, 1993): the time in seconds needed for a subject to read a given type expression and to conceive and typewrite an instance of an object with exactly this type in the 'standard' programming environment. The time between showing the type expression on 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.

The strategy for data collection is that of controlled experiments, as opposed to for example field studies. Controlled experiments have been chosen to have a better control over the instances of type expressions, and to have better control over the conditions under which the comprehensibility is measured. If a vector of measures is used, one has to compare the relative merit of each measure. This is not carried out in this exploratory study.

8.2.4 Validities in the generative phase

The validities in the generative phase are *valuation* validities: establishing the 'value' of elements, relations and embedding systems in each domain. In all domains there are validation criteria, which may be mutually conflicting. They are all desirable, but they cannot be maximised at the same time (Brinberg & McGrath, 1985).

Valuation Validition in the Substantive Domain

Three general criteria for values in the substantive domain are: the *effectiveness*, the *cost* and the *quality*. The validity of the chosen phenomena and patterns have to be considered: e.g., the value of documentation in software development; the value of type declarations in software documentation; the value of comprehensibility of type expressions. Furthermore: what is the expected improvement of the software quality by the use of good documentation; what is the cost of good documentation; what is the value and the cost of quantitative assessment of documentation quality. Finally, what is the 'value' of the chosen context with restrictions on the real world to obtain the real world model. On one hand there are restrictions on the documentation: software documentation

- formal documentation - type declarations - simple type expressions; on the other hand on the programmers: academia - novice programmers.

Valuation Validition in the Conceptual Domain

The three criteria in the conceptual domain are: *parsimony*, the use of fewer concepts and fewer relations in the interpretation of the problem; *scope*, the range of the problem being covered by the concepts (content validity); and *differentiation*, the amount of detail of the problem that can be interpreted with the concepts (construct validity). A prerequisite value is the consistency of the concepts and relations.

Even in a small scale case study as presented in this chapter, there are many concepts introduced and used: from the representational measurement theory, from programming theory to describe type expressions and the abstraction of type expression with the containment relation and the metric function. A size metric instead of the structure metric would probably require fewer concepts. The formal validation comprises the check on consistency of the concepts used. The scope and differentiation of the concepts are apparent in the given mapping rules from the real world model to the abstractions.

Valuation Validition in the Methodological Domain

In the methodological domain, the three mutually conflicting criteria are: *precision*, i.e. the accuracy of the measurement and the amount of control of the variables; *realism* of the context in which the information is obtained in relation to which that information is intended to apply or to be used; and *generalisability* with respect to the chosen entities and attributes in the problem. The chosen research strategy has to result in reliable data on the phenomena. In this exploratory study, controlled experiments have been chosen. If data are to be applied, e.g. in metric tools in industrial practice, field studies will be required. The value of a measure has to be established by comparison with other measures (the criterion validity). Only one measure for the comprehensibility has been used in the case study. The realism of the context in this study can be traced back from the given abstractions and restrictions on the real world.

The analysis of data will be derived from the axioms that have been stated in the conceptual domain in the previous section. In the executive phase, it has to be established whether or not comprehensibility of type expressions can be described in a consistent relational structure, in order to resolve the scale of measurement and to establish the correspondence with relations in the conceptual domain.

8.3 The executive phase

The first step in this phase is the collection of quantitative data for the observed phenomena as described in the previous section. The measure will be used as a criterion for the empirical relation between the entities in the given context. Subsequently, the data obtained with this measure will be analysed and the correspondence will be established with the relations in the conceptual domain. The section will be concluded will a discussion of the correspondence validities in this phase.

The experiments and the empirical order have been described in Chapter 7.4. The following approaches in the analysis of the data have been used. Firstly, a global analysis has been given based on the average time measured for each type expression (Chapter 7.4.1). Secondly, an axiomatic analysis of the preference of each subject between pairs of type expressions has been described with a test on intransitive group preferences (Chapter 7.4.2.1). Finally, an axiomatic analysis based on the relative frequencies of these preferences has been considered with a test on stochastic transitivity (Chapter 7.4.2.2). In the same section, measurement errors have been treated with a threshold probability and semiorders.

From this 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.

8.3.1 Calibration

For each of the type expressions in the data set, an expression can be derived from the metric function *m*, as defined in section 8.2.2.2. For example, the metric value for the abstraction of the type expression of the function *split* yields the expression

$$1 \times c_T + 2 \times c_F + 3 \times c_L + 4 \times c_N + 1 \times c_B + 0 \times c_C$$

This expression can be equated to the average measured time for the correct responses. With linear regression analysis of these equations for each type expression in the experiment, the calibration of the constants c_T, c_F, c_L, c_N, c_B, c_C has been obtained (cf. Chapter 6.4.3).

8.3.2 Prediction

A second data set has been obtained with subjects different from the first set, and with different type expressions. The calibrated metric function of the pre-

vious section is used to calculate the time for each type expression. The Pearson product-moment correlation coefficient (Guilford & Fruchter, 1978) between the measured values and the calculated values is 0.80. The related forecasting efficiency is 40%; i.e. a reduction in variance of the predicted comprehensibility is achieved by using the calculated metric value. It is also possible to compare the ranks of the measured and calculated values. The Spearman rank correlation coefficient is 0.74. From these results it can be concluded that there is a reasonable good agreement between the measured and predicted values in the experiment (cf. Chapter 6.4.3).

8.3.3 Discussion

The comprehensibility of simple type expressions has been operationalized as a time measurement. The ranking of the average time is in agreement with a simple extension of the partial order obtained for the corresponding abstract type expressions, despite a rather large standard deviation in the measured values, as has been described in Chapter 7.4. Axiomatic analysis has been used to localise inconsistencies in the experimental data: e.g. 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. Calibration of the metric function, as defined for abstract type expressions, has been carried out with standard regression analysis. The prediction of the comprehensibility with this calibrated metric function shows a good agreement with the measured values from an independent data set.

8.3.4 Validities in the executive phase

In the executive phase, the validity of the *correspondence* between the different relational structures has to be established: the correspondence validities. Moreover, the experimental design used in the collection of data has to be validated (design validity). The main emphasis of the case study has been on the correspondence between the relational structures as given in Figure 8.1. The correspondence between the empirical relational structure with the comprehensibility of type expressions and the numerical relational structure of the time measurement has been established. This correspondence has been validated by examining the representation axioms from measurement theory. Furthermore, the correspondence between the two numerical relational structures

has been validated by the calibration and prediction with the metric function. From this analysis based on standard statistical techniques, it has been concluded that there is a good correspondence between the empirical relational structure and the formal relational structure with abstract type expressions and the containment relation. The correspondence of real world software documentation with the comprehensibility of simple type expressions has not been validated in the case study. This will be considered in the following phase.

8.4 The interpretative phase

In this follow-up phase, the third phase in the development process, the set of findings obtained in the executive phase are interpreted. Furthermore, the repeatability of the findings, the range of variation of elements and relations (from each of the domains) over which the set of findings holds, and boundaries beyond the set of findings do not hold, are explored.

The analysis in this case study has been restricted in many ways: the subjects in the experiment (novice programmers), and the type expressions (no grouping brackets, no type variables, no type synonyms). Furthermore, alternatives of the abstraction function are not considered. Moreover, only one comprehensibility measure has been used. This leads to questions of generalisation validities.

8.4.1 Validities in the interpretative phase

The validities in the interpretative phase are *generalisation* or robustness validities. For each domain this validity is the extent to which the scope and limits of a set of empirical findings can be specified with respect to the elements and relations in that domain. Generalisation validities for each domain addresses the following aspects. *Replication*: would the same set of findings occur if the study is repeated with the same set of elements and relations? *Convergence*: would the same set of findings occur if certain facets of elements and relations are varied systematically? *Differentiation* or boundary search: if a different set of findings occurs with certain facets of elements and relations varied systematically, can these differences be explained with the relational system? It is not only important to look for the conditions under which the findings will fit the hypothesis, the invariance, but also try to identify and explain the conditions under which the findings disconfirm the hypothesis, the failures of invariance's.

To give some examples: If another measure for comprehensibility had been used, would the same order be found? Would a size metric instead of the struc-

ture metric yield a good correspondence with comprehensibility? Is there an influence of the recognition of the type declaration of often used standard functions? What is the influence of programming proficiency on the order of type expressions: novices versus experts?

Another important aspect is the generalisation of the approach outlined in this chapter 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 Fenton, 1991). A containment based order has been defined by Melton (Melton *et al.*, 1990; Fenton, 1992), and a formal axiomatic validation is presented by Zuse (1992). An experimental axiomatic validation could be carried out along the framework described in this chapter, e.g. for maintainability and structural properties.

There is also a questioning of the conceptual paradigm chosen in this chapter: representational measurement theory. Although this approach has a wide adherence in especially natural science, there are also meta-theoretical limitations, among others the absence of criteria to chose between alternative representations (Roberts, 1979). Another critical observation is made by Guttman:

> *There is much to be learned from exploring axioms and their formal consequences. But there remains the danger of seeking data merely to fit axioms. (Guttman, 1971; cited in Schwager, 1988).*

8.5 Relation with other validation approaches

There are two types of conclusions: firstly, on the topic of the case study itself, i.e. the comprehensibility of type expressions; secondly, on the validation approach as exemplified by the case study. In regard to the first point: some conclusions have been given in the discussion section of the executive phase and in the interpretative phase. As has been described in Chapter 10, the main point is the role of representation axioms in the diagnostic testing (Luce, 1990) of the empirical order of the comprehensibility of type expressions. Inconsistencies can be localised. They may hint at anomalies in the experiment or weaknesses in the theory: they can be used in the development of the conceptual domain, e.g. in the choice of alternative abstractions of type expressions.

Other approaches to the validation of software metrics can be found in the literature. Gustafson *et al.* (1992) present a classification of validation studies of software metrics. In their view, validation checks the predictive abilities of a measure against a dependent variable, while verification checks the reasonableness of the measure. For each approach it is indicated: whether verifica-

tion or validation is achieved, whether the approach produces a dependent variable, whether there is an underlying theory used or produced, and whether the results of the approach are generalizable to other data or environments. They distinguish the following (not disjunct) approaches: 1. The shotgun approach: using statistical correlation techniques between many measures. 2. The standard dependent variable approach: based on a theory with data from completed projects. 3. The controlled-experiments approach: based on a theory with data from experiments. 4. The verification approach: using formal properties of measures. 5. The exploratory approach: in which a large set of measurements is grouped with factor analysis. 6. The intuitive approach: in which measurements are correlated with judgement of experts. 7. The goal-oriented approach: in which a particular property has to be optimised. 8. De facto approaches, which fail to be classified in one of the previous categories. They conclude that the lack of planning for validation in the development of measures will result in measures that have limited usefulness and questionable validity.

Schneidewind (1992) presents a methodology for validating software metrics, from the point of view of the metric user. He discusses six validity criteria: 1. Association: the extent to which a variation in a software attribute is explained by the measure. 2. Consistency: the strength of the rank correlation between a software attribute and a measure. 3. Discriminative power: the strength of classification of a software attribute with a measure. 4. Tracking: a monotonic relation between attribute and measurement. 5. Predictability: the accuracy of predicting an attribute with a measure. 6. Repeatability: the success rate of validating the measure for an attribute. The six criteria support the three functions of measurement: assessment, control and prediction. The criteria provide a rationale for the validation, the selection and application of metrics.

As compared with these approaches, in this chapter the emphasis is on the validation of representation axioms in the different phases of the software metric development process, both formally and empirically. This approach is especially useful in a domain with a weak theoretical foundation. Validities have been differentiated in the validity network scheme. The criteria listed by Schneidewind can be found in this network. The development process is usually not a linear process, but will be iterated in a spiral development. The approaches distinguished by Gustafson *et al.* (1992) may have their own merits in different phases of this spiral process. A standard on validation issues in software measurement is urgently required (cf. American Psychological Association, 1954).