When a software measure is not a measure

by Norman Fenton

A recent interesting paper by Melton *et al.* [1] discussed finding measures which preserve intuitive orderings on software documents. Informally, if $\preceq$ is such an ordering, then they argue that a measure M is a real-valued function defined on documents such that $M(F) \leqslant M(F')$ whenever $F \preceq F'$. However, in measurement theory, this is only a necessary condition for a measure M. The representation condition for measurement additionally requires the converse; that $F \preceq F'$ whenever $M(F) \leqslant M(F')$. Using the measurement theory definition of a measure, we show that Melton *et al.*'s examples, like McCabe's cyclomatic complexity [2], are *not* measures of the proposed intuitive document ordering after all. However, by dropping the restriction to real-valued functions, we show that it *is* possible to define a measure which characterises Melton *et al.*'s order relation; this provides a considerable strengthening of the results in Reference 1. More generally, we show that there is no single real-valued measure which can characterise any intuitive notion of 'complexity' of programs. The power of measurement theory is further illustrated in a critical analysis of some recent work by Weyuker [3] *et al.* on axioms for software complexity measures.

1 Introduction

Recently, there have been a number of attempts to introduce some much needed rigour into the field of software measurement. These divide roughly between work concerned with finding axioms for measures [1, 3, 4–7] and work concerned with applying measurement theory principles to software measurement [8–11]. A common theme of this work is the emphasis on reasoning about necessary properties of measures. This is an important shift from traditional work on software measurement, which concentrated on proposing specific 'metrics' without any real thought for what these were supposed to be measuring. The traditional work also concentrated on so-called validation studies, where proposed metrics were compared with various types of project data in the hope of finding correlations. The scientific shortcomings of such studies were examined in detail in Reference 12.

We believe that the axiomatic approaches to software measurement can be greatly improved by consideration of measurement theory [13, 14]. Thus, it is envisaged that all the formal approaches could be rationalised within a measurement theory context. This paper concentrates on one major example of the axiomatic-type approach; this is the recent paper by Melton *et al.* [1].

We describe how Melton *et al.* were looking for measures which characterised a specific view of program complexity, formalised in terms of a special flowgraph order relation. It is shown that the proposed requirements for such measures are incomplete because the so-called representation condition for measurement is only partially satisfied. Consequently, it is shown that their example 'measures' are not measures in the sense of measurement theory. In fact, it is shown that there is no possible real-valued measure which preserves the stated order relation. However, we show that it is possible to construct a measure which preserves the order relation, but which is not 'real-valued' in the usual sense.

A by-product of these results is that it is impossible to construct any single real-valued measure which captures any general notion of program complexity. This brings into question the theoretical validity of much work in the software 'metrics' area. In the light of this, Weyuker's axioms [3] for software complexity measures are analysed critically. By a straightforward application of measurement theory, a major inconsistency is identified. A simple application of measurement theory also shows how a recent critique of Weyuker's axioms [15] is itself flawed.

2 Order-preserving measures and measurement theory

For several years, researchers have attempted to define software 'complexity' measures which are supposed to capture intuitive notions of complexity, including cognitive notions, and which are supposed to be indicative of such varying product attributes as reliability, maintainability, and

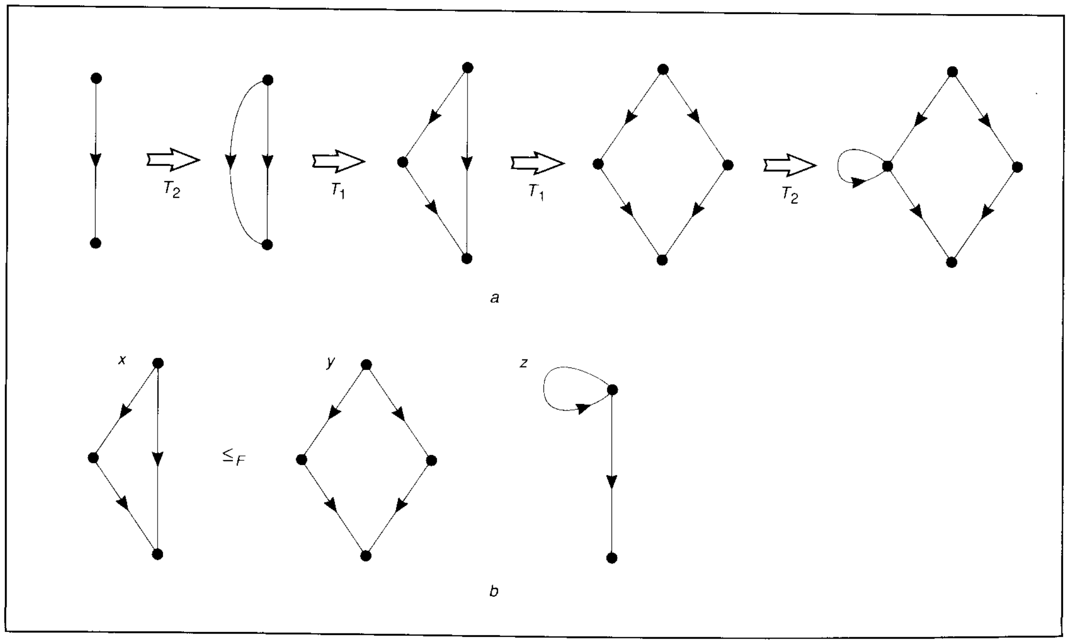

Fig. 1 The relation $\leqslant_F$

testability [2, 16–19]. Melton *et al.* [1] have identified some of the problems associated with such approaches. As an alternative, they proposed that we must first concentrate on studying properties of the objects of measurement. In this case, the objects are software *documents*, an important example of which is the flowgraph model of program structure.

Melton *et al.* argue that there are intuitive, but formally definable, orderings on flowgraphs which characterise specific notions of structural complexity. Informally, if $\preceq$ is such an ordering, then they argue that a measure M is a real-valued function defined on flowgraphs such that $M(F) \leqslant M(F')$ whenever $F \preceq F'$.

The approach of Melton *et al.* is, in fact, subsumed within the representational theory of measurement, as described in References 13, 14 and 20. This is concerned with the formal requirements for measurement, and includes a comprehensive theory of measurement scales and meaningfulness in measurement. Measurement is the process by which numbers or symbols are assigned objectively to objects in such a way as to characterise some specific attribute possessed by the objects. A basic tenet of measurement theory is that, as Melton *et al.* propose, there must be some intuitive understanding of the attribute Q (such as 'complexity') of the objects to be measured (in this case programs) in advance of the numerical assignment. This intuitive understanding is characterised by empirical relations over the set C of relevant objects (or formal models of these objects). In this paper, the set C, together with the set of empirical relations R, is called an *empirical relation system* for the attribute Q. It is thus the pair (C, R) which characterises our empirical understanding of the attribute Q in C. A measure M for Q is a mapping from the empirical relation system into some numerical relation system such that the empirical relations are preserved by numerical relations and vice versa. This is the so-called *representation condition* which is the key component of measurement theory. Only when the representation condition is satisfied, can M be said to be a measure for (C, R) or of Q.

In the simplest case, which for brevity is considered here, there is just one relation R over C, such as the program flowgraph ordering. In addition, if, as in this case, R is binary, then for each pair of objects $A, B \in C$ either $(A, B) \in R$ or $(A, B) \notin R$. In these special circumstances, the representation condition amounts to the following:

A measure M for the empirical relation system (C, R) is a mapping $M: C \rightarrow N$ where N is some set of numbers or symbols with a binary relation P defined over N such that for each $x, y \in C$

$$(x, y) \in R \text{ if and only if } (M(x), M(y)) \in P \qquad (1)$$

In general, the representation condition makes no other restrictions about the set N and the relation P defined over N. This means that, contrary to popular belief, the set N does not have to be the set of real numbers *Real*, and P does not have to be a relation such as $\leqslant$. This coincides with the formal view that measurement is not always real-valued. Examples of the use of the complex numbers in electrical engineering should dispel the notion that real numbers are the only useful number system for measurement. As a software example, it is shown in Reference 21 that, with regard to measuring algorithmic efficiency, true measurement is only achieved when the restriction to the real numbers is dropped. Moreover, classification of a set of entities is a special form of measurement (called nominal measurement), and words or symbols other than numbers are normally used for this purpose. For example, software faults are often classified as *specification*, *design*, *coding* or *maintainance*, according to the life-cycle phase in which they were introduced.

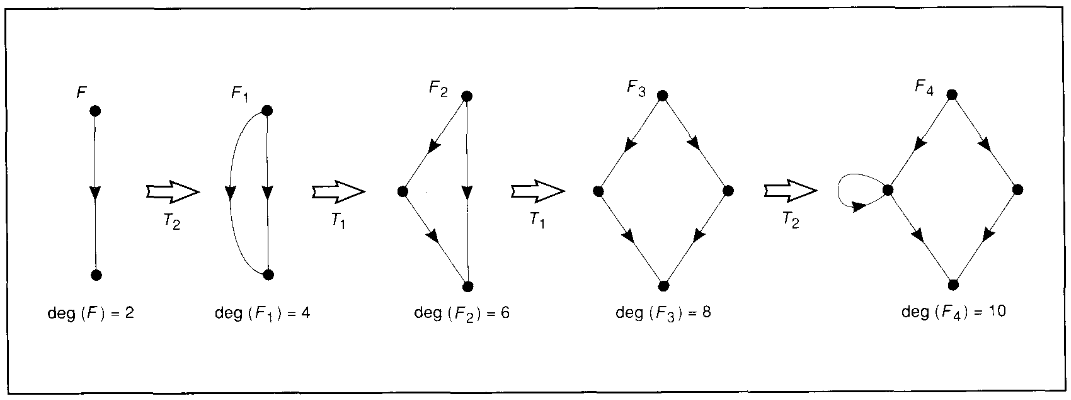

Fig. 2 The degree of a flowgraph

However, in the case where R is an 'order' relation $\preceq$, it is natural to ask if it *is* possible to find a mapping into the set *Real* such that $\preceq$ is mapped to $\leqslant$. In this case, the representation condition asserts, that for each $x, y \in C$,

$$x \preceq y \text{ if and only if } M(x) \leqslant M(y) \tag{2}$$

This can now be compared to the definition of a software measure given in Reference 1, which only requires the *necessity* in eqn. 2. Specifically, in Reference 1, M does not necessarily satisfy

$$M(x) \leqslant M(y) \Rightarrow x \preceq y \tag{3}$$

2.1 The special order relation $\preceq_F$

In the following, we shall assume that a flowgraph is, as usual, a digraph with distinguished start and stop nodes. Every node lies on a walk from the start to the stop node, and the stop node has outdegree 0.

The particular order relation $\preceq_F$ in Reference 1 is defined on the set $\mathscr{F}$ of all flowgraphs in the following way:

$x \preceq_F y$ if y can be derived from x by one or more transformations of the form T_1 or T_2.

Informally, a T_1 transformation is the insertion of a node into an existing flowgraph arc, whereas a T_2 transformation is the insertion of a new arc between two existing nodes. This is illustrated in Fig. 1*a*. Thus, for example, for the flowgraphs x, y and z in Fig. 1*b*, it is easily seen that

$$x \preceq_F y, \text{ but neither } x \preceq_F z \text{ nor } z \preceq_F y \tag{4}$$

The ordering $\preceq_F$ characterises an intuitive notion of structural complexity over $\mathscr{F}$. However, because of eqn. 4, it follows that there is no real-valued measure M for the intuitive notion of structuredness which is characterised by $\preceq_F$. For if there was, then we would deduce that $M(x) \leqslant M(y)$ while at the same time $M(x) > M(z)$ and $M(z) > M(y)$ (and hence $M(x) > M(y)$, a contradiction). Thus, formally we may assert the following.

Theorem 1

There is no measure M for the empirical relation system $(\mathscr{F}, \preceq_F)$ in the numerical relation system (Real, $\leqslant$).

It is instructive to show how eqn. 3 fails for one of the examples of an 'order preserving software measure' given in Reference 1. This is McCabe's cyclomatic complexity number [2]. Let x, y and z be as in Fig. 1*b*. If M is cyclomatic complexity, then

$$M(x) = 2,\ M(y) = 3,\ M(z) = 2$$

Thus, $M(z) \leqslant M(y)$ but it is not the case that $z \preceq_F y$. Hence, although M is order-preserving in the sense of Reference 1, it is not a measure in the sense of measurement theory.

Another way of understanding why there are no real-valued measures for $\preceq_F$ is to note that, whereas $\leqslant$ is a linear order, $\preceq_F$ is only a partial order. Flowgraphs which are incomparable with respect to $\preceq_F$, such as y and z above, have an order 'forced onto them' by any mapping into *Real*.

Therefore, if there is no single real-valued measure for the view of complexity which is characterised by the relation $\preceq_F$, what hope is there of finding a single real-valued measure for even more general notions of program complexity? The answer is definitive; none whatsoever. In fact, the counter-example provided by x, y and z in Fig. 1*b* would probably work for any general notion of complexity. However, this negative result does not mean that there are no interesting program measures. Instead, it reinforces the point made repeatedly in Reference 10 that we have to restrict our attention to measuring very specific attributes which contribute toward complexity. For example, the maximum depth of nesting, structuredness based on the distribution of primes in the decomposition tree [5, 22], and the number of paths of various types can all be measured rigorously and automatically.

Before constructing a (non real-valued) measure for the empirical relation system $(\mathscr{F}, \preceq_F)$, it is worth noting that there are a number of theorems used in measurement theory that assert conditions under which certain scales of direct measurement are possible for certain relation systems. For theorems that describe the conditions under which an empirical relation system has an ordinal, interval or ratio scale representation, see References 14 and 20.

3 A measure which does preserve $(\mathscr{F}, \preceq_F)$

Since it is impossible to find a measure in the numerical relation system (*Real*, $\leqslant$), it is necessary to consider a

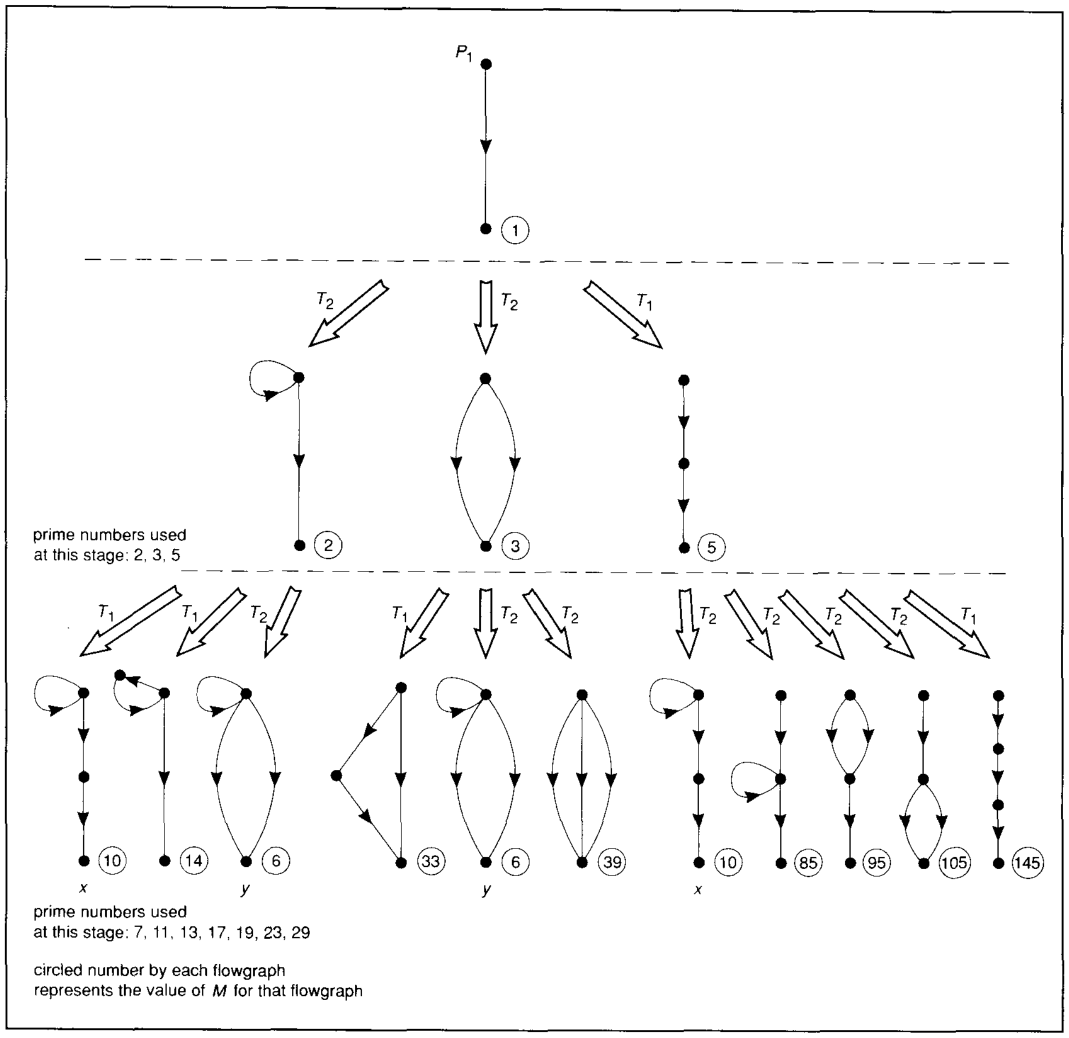
P_1
1
T_2
T_2
T_1
prime numbers used at this stage: 2, 3, 5
2
3
5
T_1
T_1
T_2
T_1
T_2
T_2
T_2
T_2
T_2
T_2
T_1
10
14
6
33
6
39
10
85
95
105
145
x
y
y
x
prime numbers used at this stage: 7, 11, 13, 17, 19, 23, 29
circled number by each flowgraph represents the value of M for that flowgraph

Fig. 3 The first three stages in the contruction of a measure for ($\mathscr{F}$, $\leqslant_F$)

different numerical relation system. The one chosen is (*Nat*, |) where *Nat* is the set of natural numbers and | is the relation 'divides without remainder'. Thus, 3|6 and 7|21, but it is not the case that 7|22. It is intended to prove the following constructively.

Theorem 2

There is a measure $M : (\mathscr{F}, \leqslant_F) \rightarrow (\mathscr{N}, |)$.

Specifically what must be found is a mapping M which satisfies the representation condition, i.e.

$$F \preceq_F F' \text{ if and only if } M(F) \mid M(F') \qquad (5)$$

First, a Lemma, illustrated in Fig. 2, is needed.

Lemma 1

Let the degree of F, written deg(F), denote the sum of the vertex degrees in a flowgraph F. Applying a single transformation (either T_1 or T_2) to F always yields a flowgraph of degree deg(F) + 2.

It follows that the degree is an invariant property of all the flowgraphs which can be derived from the given flowgraph F. We shall also need the following Lemma, whose proof is given in the Appendix.

Lemma 2

Let F be a flowgraph with at least two edges. Then F is generated by one of T_1 or T_2.

It also follows that every flowgraph has even degree and that every flowgraph of degree $n + 2$ is derived from a flowgraph of degree n by a single transformation. Now Theorem 2 can be proved by induction on n, the degree of a flowgraph. Specifically, a measure M is constructed which satisfies eqn. 5 for flowgraphs of degree n for each even $n \geqslant 2$ (the process is illustrated in Fig. 3).

The only flowgraph of degree $n = 2$ is the flowgraph P_1 illustrated in Fig. 3. Define $M(P_1) = 1$. Trivially, eqn. 5 is satisfied by all pairs of flowgraphs of degree $n \leqslant 2$. Therefore, next consider $n \geqslant 4$. Inductively assume that, for every flowgraph F of degree $\leqslant n$, $M(F)$ has been defined in such a way that eqn. 5 is satisfied. Then we have to show how to define $M(F)$ for each flowgraph of degree $n+2$ in such a way that eqn. 5 is satisfied.

For each flowgraph F of degree n, consider the set $S(F)$ of flowgraphs which can be derived from F by a single transformation. By Lemma 2, each flowgraph of degree $n+2$ must be in at least one of these sets $S(F)$. The problem is that a flowgraph F' of degree $n+2$ may be in *more* than one of these sets, i.e. it can be derived by a single transformation from *different* flowgraphs F_1 and F_2. For example, in Fig. 3 the flowgraph marked x appears twice in the derivation tree because it is derived from different flowgraphs by different transformations. The same is true of the flowgraph marked y. In such cases, we have to ensure that both $M(F_1) \mid M(F')$ and $M(F_2) \mid M(F')$. Therefore, what we do is first consider each such 'duplicate' flowgraph F'. Let $F_i, \ldots, F_j$ be the collection of flowgraphs of degree n from which F' may be derived. Then we define

$$M(F') = M(F_i) \times \cdots \times M(F_j)$$

This ensures the required divisibility relations.

Having dealt with all the duplicate flowgraphs, all that remains are those flowgraphs which are derivable from a unique flowgraph of degree n. Suppose $F'_1, \ldots, F'_k$ is this set of flowgraphs, and suppose that these are derived from $F_1, \ldots, F_k$, respectively. Then choose k distinct prime numbers $p_1, \ldots, p_k$ not already used in the definition of any $M(F)$. Then define

$$M(F'_i) = p_i \times M(F_i) \text{ for each } i$$

By definition, eqn. 5 is preserved for all flowgraphs of degree $n+2$, and so it follows by induction (and Lemma 3) that M satisfies eqn. 5 for all flowgraphs. Hence, Theorem 2 is proved. Fig. 3 illustrates the actual values of M for the smallest 15 flowgraphs.

It is important to make one final observation about the order relation $\preceq_F$, which suggests that it is a very weak characterisation of structural complexity. Suppose F_1 and F_2 are flowgraphs. Then these may be concatenated, using the normal sequence operation [5], to form the flowgraph $(F_1; F_2)$. Any reasonable notion of flowgraph ordering $\preceq$ ought to yield $F_1 \preceq (F_1; F_2)$ and $F_2 \preceq (F_1; F_2)$. In fact, neither of these is true of $\preceq_F$; it is not possible to transform F_1 to $(F_1; F_2)$ by a finite sequence of transformations of type T_1 and T_2.

4 Axiomatising complexity?

In Section 2, we showed that attempts to define general software 'complexity' measures were doomed to failure. It is counter-productive to insist on equating measures of specific (and often important) structural attributes with the poorly understood attribute of complexity. Yet, it is widely believed that such measures can have the magical properties of being 'indicators' of such diverse notions as *comprehensibility*, *correctness*, *maintainability*, *reliability*, *testability* and *ease of implementation*. A high value for a 'complexity' measure is supposed to be indicative of low comprehensability, low reliability etc. Sometimes (rather ironically) these measures are also called 'quality' measures [17]. In this case, high values of the measure actually indicate low values of the quality attributes.

The danger of attempting to find measures which characterise so many different attributes is that we inevitably find that they have to satisfy *conflicting* aims. An important example of this is found in Reference 3, where Weyuker lists a number of properties which she believes any complexity measure M must satisfy if it is to conform to generally accepted expectations. Two of the properties are

- *Property A:* for any programs P, Q, $M(P) \leqslant M(P; Q)$ and $M(Q) \leqslant M(P; Q)$ (adding code to a program can only increase its complexity).
- *Property B:* there are programs P, Q and R such that $M(P) = M(Q)$ and $M(P; R) \neq M(Q; R)$ (we can find two programs of equal complexity which, when separately concatenated to a same third program, yield programs of different complexity).

Property A is reasonable for any view of complexity which is related to program *size*. However, if complexity is related to low comprehensibility, then Property A is unreasonable since our general level of comprehension of a program may *increase* as we see more of it. Therefore, we confidently conclude from Property A that Weyuker's notion of complexity emphasizes size. On the other hand, Property B has much to do with comprehensability and little to do with size.

It follows that properties A and B are relevant for very different, and incompatible, views of complexity. It is impossible to define a set of consistent axioms for a completely general view of 'complexity'. It is far better to concentrate, as we have proposed, on specific attributes and consider 'axioms' for measures of these. This is the true measurement theory approach. Unfortunately, measurement theory, and in particular the representation condition which would have simplified much of the work, is totally ignored in Reference 3.

The only valuable lesson to be drawn from Weyuker's properties is the confirmation that the search for general complexity measures is doomed to failure. However, the general misunderstanding of scientific measurement in software engineering is illustrated further in a recent paper [15] which has criticised Weyuker's axioms for the wrong reasons. Cherniavsky and Smith [15] define a code-based 'metric' which satisfies all of Weyuker's axioms but, which they rightly claim, is not a sensible measure of complexity. They conclude that axiomatic approaches may not work. There is no justification for their conclusion. On the one hand, as they readily accept, there was no suggestion that Weyuker's axioms were complete. More importantly, what they fail to observe is that Weyuker did not propose that the axioms were *sufficient*; she only proposed that they were necessary. Since the Cherniavsky/Smith 'metric' is clearly not a measure (in our sense) of any specific attribute, then showing that it satisfies any set of necessary axioms for any measure is of no interest at all.

These problems would have been avoided by a simple lesson from measurement theory. The definition of a numerical mapping does not in itself constitute measurement. It is popular in software engineering to use the word 'metric' for any number extracted from a software entity. Thus, although every measure is a 'metric', the converse is certainly not true. The confusion in References 3 and 15 arises from wrongly equating these two concepts.

5 Conclusions

Perhaps the most fundamental lesson to be learnt from measurement theory is that some kind of intuitive understanding of an attribute necessarily precedes its measurement. The intuitive understanding is normally characterised by empirical relations and axioms.

The work of Melton *et al.* [1] attempts to formalise intuitive understanding of a specific view of program complexity in terms of an empirical order relation $\preceq$. They assert that any measure M of program complexity must preserve this relation, in the sense that $F \preceq F'$ implies $M(F) \leqslant M(F')$. This approach is consistent with measurement theory. However, the representation condition of measurement asserts additionally that a measure cannot create any new comparabilities. Thus, $M(F) \leqslant M(F')$ must also imply that $F \preceq F'$. Taking this into consideration, there is no real-valued measure which characterises this view of complexity. However, it is possible to construct a non-real-valued measure. The fact that it is of little practical value is more an indictment on the view of complexity characterised by $\preceq$.

Much work in software measurement has ignored a basic measurement principle; that you must know what you are measuring before you propose a numerical mapping. Some simple lessons in measurement theory are enough to confirm that it is pointless to attempt to define a real-valued measure which captures *different* intuitive attributes. Yet, this is precisely what many people have been attempting in software metrics for many years.

6 Acknowledgments

The author would like to thank Robin Whitty for his incisive comments and, in particular, for the proof of Lemma 3; the referees, whose comments helped to significantly improve the paper; and David Wright and Bev Littlewood, whose useful comments on the initial draft also lead to improvements.

The author is supported in part by NATO Collaborative Research Grant 034/88 and by ESPRIT2 Project PDCS.

7 References

[1] MELTON, A.C., GUSTAFSON, D.A., BIEMAN, J.M., and BAKER, A.A.: 'A mathematical perspective of software measures research', *Softw. Eng. J.*, 1990, **5**, (5), pp. 246–254

[2] McCABE, T.J.: 'A complexity measure', *IEEE Trans.*, 1976, **SE-2**, (4), pp. 308–320

[3] WEYUKER, E.J.: 'Evaluating software complexity measures', *IEEE Trans.*, 1988, **SE-14**, (9), pp. 1357–1365

[4] DARÓCZY, Z., and VARGA, L.: 'A new approach to defining software complexity measures', *Acta Cybern.*, 1988, **8**, (3), pp. 287–291

[5] FENTON, N.E., and WHITTY, R.W.: 'Axiomatic approach to software metrication through program decomposition', *Comput. J.*, 1986, **29**, (4), pp. 329–339

[6] PRATHER, R.E.: 'An axiomatic theory of software complexity measure', *Comput. J.*, 1984, **27**, pp. 340–347

[7] SHEPPERD, M., and INCE, D.: 'Algebraic validation of software metric'. European Software Engineering Conf., Toulouse, France, 1991

[8] BAKER, A., BIEMAN, J., FENTON, N.E., GUSTAFSON, D., MELTON, A., and WHITTY, R.W.: 'A philosophy for software measurement', *J. Syst. Softw.*, 1990, **12**, pp. 277–281

[9] DeMILLO, R.A., and LIPTON, R.J.: 'Software project forecasting' *in* PERLIS, A.J., SAYWARD, F.G., and SHAW, M. (Eds.): 'Software metrics' (MIT Press, 1981) pp. 77–89

[10] FENTON, N.E.: 'Software metrics: a rigorous approach' (Chapman & Hall, London, 1991)

[11] ZUSE, H.: 'Software complexity: measures and methods' (de Gruyter, 1990)

[12] FENTON, N.E., and KITCHENHAM, B.A.: 'Validating software measures', *J. Softw. Test. Verif. Reliab.*, 1991, **1**, (2), pp. 27–42

[13] FINKELSTEIN, L.: 'A review of the fundamental concepts of measurement', *Measurement*, 1984, **2**, (1), pp. 25–34

[14] ROBERTS, F.S.: 'Measurement theory with applications to decision making, utility, and the social sciences' (Addison Wesley, 1979)

[15] CHERNIAVSKY, J.C., and SMITH, C.H.: 'On Weyuker's axioms for software complexity measures', *IEEE Trans.*, 1991, **SE-17**, (6), pp. 636–638

[16] CHAPIN, N.: 'A measure of software complexity'. Proc. NCC, 1979, pp. 995–1002

[17] KAFURA, D., and HENRY, S.: 'Software quality metrics based on interconnectivity', *J. Syst. Softw.*, 1981, **2**, pp. 121–131

[18] OVIEDO, E.I.: 'Control flow, data flow, and program complexity'. Proc. COMPSAC, New York, 1980 (IEEE Computer Society Press) pp. 146–152

[19] STETTER, F.: 'A measure of program complexity', *Comput. Lang.*, 1984, **9**, (3), pp. 203–210

[20] KRANTZ, D.H., LUCE, R.D., SUPPES, P., and TVERSKY, A.: 'Foundations of measurement' (Academic Press, 1971) Vol. 1

[21] FENTON, N.E.: 'The mathematics of complexity and measurement in computer science and software engineering' *in* JOHNSON, J., and LOOMES, M. (Eds.): 'The mathematical revolution inspired by computing' (Oxford University Press, 1991), pp. 243–256.

[22] PRATHER, R.E., and GIULIERI, S.G.: 'Decomposition of flowchart schemata', *Comput. J.*, 1981, **24**, (3), pp. 258–262

[23] MADER, W.: 'Connectivity and edge-connectivity in finite graphs' *in* BOLLOBAS, B. (Ed.): 'Surveys in combinatorics' (Cambridge University Press, 1976), pp. 66–95

The author is with the Centre for Software Reliability, City University, London EC1V 0HB.

The paper was first received on 2 October 1991 and in revised form on 22 April 1992.

8 Appendix

Proof of Lemma 3

Suppose F is not generated by T_1. We shall show that it is generated by T_2. Suppose that F has start node a and stop node z. Construct a new graph $F*$ from F as follows: first add an edge from z to a. If a now has indegree 1 and outdegree 1, then contract the edge leading out of a. The resulting graph $F*$ is strongly connected, and the only node of $F*$ which may have indegree and outdegree 1 is z; otherwise F could be generated by T_1. Now it is impossible to generate F by T_2 if, and only if, there is no edge in $F*$ whose deletion leaves a strongly connected graph. By Theorem 20 in Reference 23, such a graph must have at least two nodes of indegree 1 and outdegree 1, which is a contradiction. Therefore, F must be generated by T_2 as required.