Axiomatic approach to Software Metrication through Program Decomposition

N. E. FENTON*
Department of Electrical and Electronic Engineering, Polytechnic of the South Bank, Borough Road, London SE1 0AA

R. W. WHITTY
Department of Mathematical Sciences, University of London Goldsmiths' College, New Cross, London SE14 6NW

Software metrication is a major problem in the engineering of software and software-related systems. To give coherence to attempts at metrication and to allow convincing statistical validation of these attempts it is proposed to axiomatise assumptions made about software parameters and 'complexity'. The axioms are intrinsically related to the 'decomposition hierarchy' of programs, and this important notion is formalised in this paper for arbitrary programs.

Received August 1985

1. INTRODUCTION

The need for software complexity metrics is well documented in the literature and some excellent recent approaches may be found.[2, 8, 12] In this paper we are interested in the definitions and derivations of the metrics themselves. The term 'software complexity metric' is itself rather misleading – flattering even – since it is usually used in the software community to refer to any parameter (or explicitly a non-negative real-valued function) defined on software (or explicitly a suitable representation of the software) which it is *hoped* is some indicator of one or more aspects of the software's complexity. A problem with many previous approaches to metrication is the lack of a sound theoretical basis; this has often led to (*a*) ill-defined functions purporting to be metrics or (*b*) functions which rather than reflecting any genuine inherent complexity are at best crude 'predictors' of (already) measurable software features like number of bugs; in this sense they are usually no better (and in some cases worse) than a simple 'lines of code count'.[1, 6] In this paper we hope to redress this unfortunate balance.

A recent paper by Prather[8] has attempted to characterise the common properties which software complexity metrics ought to satisfy by describing them in the context of an axiomatic theory. This is a highly commendable approach and one which is taken considerably further in this paper. The most serious limitation of Prather's axioms is that they are defined only for the class of 'structured' programs, where structuredness corresponds informally to the usual notion of *D*-structuredness.[11] Thus in order to extend metrics to arbitrary programs (i.e. possibly 'unstructured' ones) a totally different approach (i.e. different from the axiomatic approach) is taken by Prather and also incidentally by others who have defined similar metrics.[12] We consider this to be an unnecessary divergence from a unified approach; we have already argued at length[3, 4, 10, 11] that the traditional notion of *D*-structuredness as the basis for structured design and analysis is unnecessarily and artificially restrictive, and nowhere is this more apparent than when we are attempting to axiomatise properties of structure. Structuredness is most naturally defined (as in Refs. 4 and 10) in terms of arbitrary families of basic structures, so that programs do not fall into just two categories, 'structured' or 'unstructured' (as assumed by many researchers in this area), but all have a quantifiable degree of structure characterised by the hierarchy of basic structures on which it is built. In section 2 we provide a formal description of the hierarchical structure of any program. This general notion of structure is used in section 3 to consider metrication axioms and metrics themselves defined on any class of programs, thus overcoming the limitations of previous approaches. Finally in section 4 we suggest areas for further development using the axiomatic approach.

* To whom correspondence should be addressed.

2. PROGRAM STRUCTURE

The basis for many of the ideas in this section is to be found in Refs. 4, 10 and 13. Although reference will be made to these earlier works, the exposition here is self-contained and contains several new results and notations.

2.1. Definition

A *flowgraph* $F = (G, a, z)$ is a triple consisting of a directed multigraph G together with distinguished nodes a, z of G satisfying:

(i) the node a (the *start* node) has directed paths to all other nodes of G;

(ii) every node has a directed path to z (the *stop* node) which has zero outdegree.

The above definition of flowgraph allows nodes of arbitrary outdegree. In general a node of outdegree n will be called an *n-ary predicate node* ($n \geqslant 2$), while nodes of outdegree 1 are called *procedure nodes*. Certain flowgraphs occur sufficiently often to merit special names. In particular Fig. 1 describes (in the usual way) the flowgraphs P_n ($n \geqslant 0$), D_1, D_1', D_2, D_3, D_4, C_2 which will subsequently be referenced by name only. As usual the start and stop nodes are distinguished diagrammatically by encircling. It has been shown in Refs. 4 and 10 how a given program written in a procedural language is mapped on to a unique flowgraph; an example of a program (written in Pascal) and its associated flowgraph is given in Fig. 2. The model is similar to the usual flowchart representation except that in addition to the

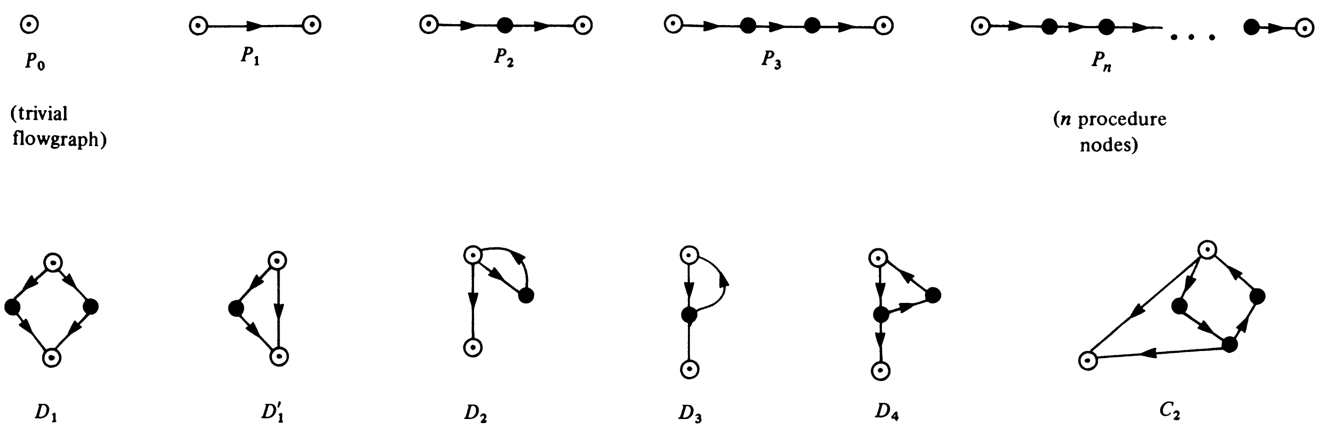


Figure 1

```
Program Example (input, output);
        {Taken from [2]}
Var x, y : real;
Begin
   read (x);
   if x = 0 then read (y)
      else
      begin
         while x < > 0 do
            read (x);
         read (y)
      end
end
```

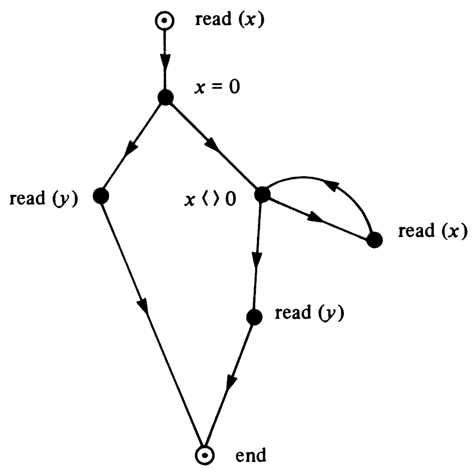


Figure 2

introduction of the necessary formality needed for rigorous analysis it removes (i) ambiguities about node association and (ii) redundancies like junction nodes and start box.

In this paper we are concerned only with complexity caused by control flow, so we shall have no need to consider *labelled* flowgraphs as we did in Ref. 4 and in particular in Ref. 3. However, future developments of this work will certainly consider both structural and linguistic characteristics (as in Ref. 3), in which case we shall simply impose the labelling on the same model.

The 'D-graphs' D_1, D'_1, D_2, D_3 (or sometimes certain subsets of these) are usually taken as the building blocks for 'structured' programs; thus in the literature the latter are usually informally defined as being those programs (i.e. flowgraphs) which can be 'built up' using these structures together with operations of 'sequence' and 'nesting'. We formalise these concepts for the more general definition of structured programs considered here.

2.2. Definition

(i) *Nesting* (*Composition*). Given a flowgraph F_1 with some procedure node x, we nest a second flowgraph F_2 on x, written $F_1(F_2$ on $x)$, by deleting the unique arc leading out of x, identifying the stop node of F_2 with the node this arc led to in F_1 and identifying the start node of F_2 with x (an example is given in Fig. 3(*a*)).

(ii) *Sequence*. Given two flowgraphs F_1 and F_2 we shall denote by seq (F_1, F_2) or more conveniently $F_1;F_2$ the flowgraph obtained by concatenating F_1, F_2 in the obvious way by identifying the stop node of F_1 with the start node of F_2 (an example is given in Fig. 3(*a*). In an obvious generalisation we may also define seq $(F_1, \ldots, F_n)$ for n flowgraphs.

The reader should note that completely formal definitions of these operations may be given by specifying the underlying digraphs (as was done in Ref. 4 for the nesting operation).

In this paper we shall attach great importance to 'depth' of nesting. Suppose that in addition to the situation in (2.2.i), y is a procedure node of F_2, and F_3 is another flowgraph. Then the flowgraph obtained by nesting F_3 on y in $F_1(F_2$ on $x)$ is the same as that obtained by first nesting F_3 on y in F_2 and nesting the resulting flowgraph on x in F_1, i.e.

$$(F_1(F_2 \text{ on } x))(F_3 \text{ on } y) = F_1((F_2(F_3 \text{ on } y)) \text{ on } x). \quad (1)$$

Thus nesting is associative, and in this case we have two levels of nesting. However, if y is (like x) a procedure node of F_1 then we could nest F_2 on x, F_3 on y simultaneously and arrive at the same result as either of the flowgraphs in (1). The latter notation suggests two levels of nesting, which does not correspond to the intuitive interpretation here, which clearly views F_2, F_3 nested at the same (first) level in F_1 in stark contrast to the previous example. Any possible confusion in this respect is avoided by using the following notation.

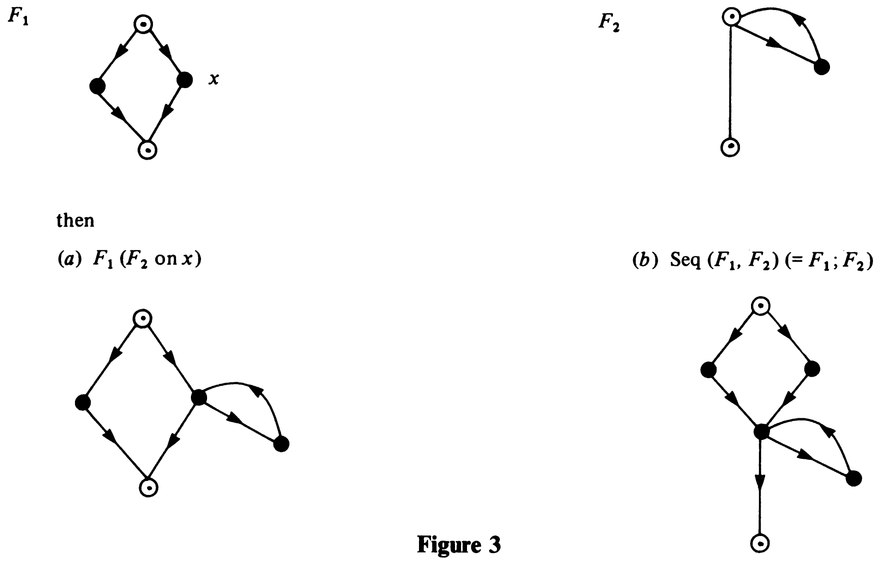


Figure 3

2.3. Definition and notation

If a flowgraph F has n (distinct) procedure nodes $x_1,\ldots,x_n$ and if $F_1,\ldots,F_n$ are arbitrary flowgraphs then the unique flowgraph formed by nesting (simultaneously) F_i on x_i ($i = 1,\ldots,n$) is denoted

$$F(F_1 \text{ on } x_i, \ldots, F_n \text{ on } x_n).$$

Thus in the previous example we would have to have written $F_1(F_2$ on x, F_3 on $y)$ instead of $(F_1(F_2$ on $x))$ $(F_3$ on $y)$, since the latter now specifically implies that y is a procedure node of F_2 and *not* of F_1.

2.4. Remark

(i) The operation of sequence may now be seen to be a special case of (simultaneous) nesting, for if x_i is the ith node of the flowgraph P_n (Fig. 1) then

$$\text{seq}(F_1,\ldots,F_n) = P_n(F_1 \text{ on } x_1,\ldots,F_n \text{ on } x_n).$$

In Ref. 4 we did, for conciseness, use just the single operation of nesting. In fact even P_n may be derived by continually nesting P_2s on to P_2s (as was done in Ref. 4 to make do with the fewest 'base' structures). Since this again suggests levels of nesting which do not intuitively exist we shall refrain from nesting any P_k ($k \geqslant 2$) on a P_n in this paper.

(ii) For any flowgraph F with procedure node x,

$$F(P_1 \text{ on } x) = F$$

and more generally, if $x_1,\ldots,x_n$ are procedure nodes of F then

$$F(P_1 \text{ on } x_1,\ldots,P_1 \text{ on } x_n) = F$$

$$F(F_1 \text{ on } x_1, P_1 \text{ on } x_2,\ldots,P_1 \text{ on } x_n) = F(F_1 \text{ on } x_1), \text{ etc.}$$

(iii) The flowgraph P_n is equal to seq $(P_1,\ldots,P_1)$ – this also follows directly from (i) and (ii) above – i.e. is just a sequence of n procedures.

2.5. Definition

A *subflowgraph** $F' = (G',a',z')$ of $F = (G,a,z)$ is a flowgraph for which G' is a subgraph of G, and the only entry into G' from $G\backslash G'$ is via a' and the only exit from G' to $G\backslash G$ is via z'.

Note that if F_2 is nested in F_1 then F_2 is a subflowgraph of F_1. Moreover, any subflowgraph can be viewed as having been obtained by nesting in this way. This leads us to the notion of *decomposition* of subflowgraphs as an inverse operation to nesting (composition). Thus the *decomposition of F_2 in F_1* (where F_2 is a subflowgraph of F_1) is the flowgraph obtained by collapsing F_2 to a single arc (x,z'), where z' is the stop node of F_2 and x is a new procedure node 'replacing' F_2. The resulting flowgraph is denoted $F_1(x$ for $F_2)$. An example is given in Fig. 4.

As in the case of nesting, we shall use an analogous

* This definition of subflowgraph corresponds to the 1-entry subflowgraph in Ref. 4. We have dropped the '1-entry' prefix here since these are the only subflowgraphs considered here.

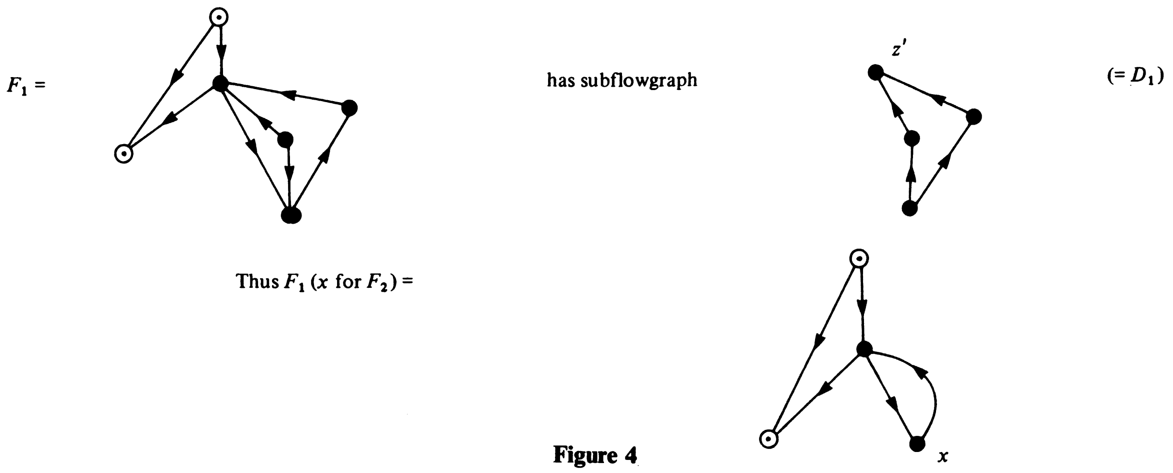
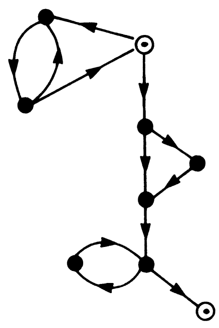
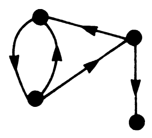
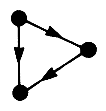


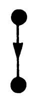
Figure 4



is sequential with components F_1, F_2, F_3, F_4 where

$F_1 =$ 

$F_2 =$  $(= D_1')$

$F_3 =$  $(= P_1$

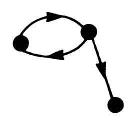
$F_4 =$  $(= D_3)$

Figure 5

notation for 'simultaneous' decomposition; explicitly, if F contains edge-disjoint subflowgraphs $F_1, \ldots, F_n$ then these may be unambiguously decomposed in F simultaneously. The resulting flowgraph is denoted

$$F(x_1 \text{ for } F_1, \ldots, x_n \text{ for } F_n).$$

We now have the necessary tools to tackle the question of flowgraph structure.

2.6. Definition

A flowgraph F will be called *sequential* if there are non-trivial subflowgraphs F_1, F_2 of F for which $F = \text{seq}(F_1, F_2)$ and *non-sequential* otherwise. It follows easily that a given sequential flowgraph F may be written *uniquely* as $F = \text{seq}(F_1, \ldots, F_k)$ $(k \geqslant 2)$ where each F_i is non-sequential. In this case the F_is are called the *components* of F. An example of a sequential flowgraph and its components is given in Fig. 5.

2.7. Definition

A flowgraph f will be called *irreducible** if it is non-sequential and contains no proper subflowgraph (except for those of type P_1).

* Because of the changed definition of subflowgraph from Ref. 4, the 'irreducibles' here are in a different class (in fact one which strictly contains that of Ref. 4.)

If a flowgraph F is non-sequential then we can unambiguously consider the set of maximal proper subflowgraphs of F $(\neq P_1)$. This set, say $\{F_1, \ldots, F_k\}$ (which will be empty if an only if F is irreducible) is uniquely defined, the F_is being edge-disjoint (again, see [4]). Thus again we shall in this case call the F_is the *components* of F (see Fig. 6 for an example).

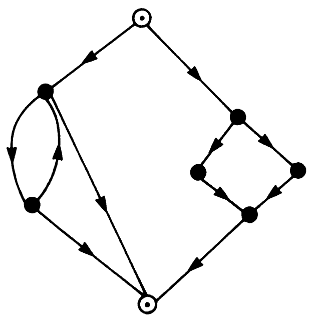
Thus any flowgraph F, which is not irreducible, has a uniquely defined non-empty set of components $\{F_1, \ldots, F_k\}$, say where $F = \text{seq}(F_1, \ldots, F_k)$ in the case where F is sequential and $F_1, \ldots, F_k$ are the maximal proper subflowgraphs of F otherwise. In what follows we shall see that a flowgraph's 'structure' is uniquely determined by recursive decomposition of its components. Before formally defining this we present the most general definition of structuredness.

2.8. Definition

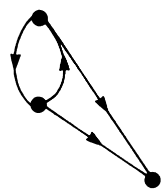
Let $\mathscr{S}$ be a family of irreducible flowgraphs. The class of $\mathscr{S}$-graphs is defined inductively by the following.

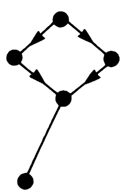
(i) Each member of $\mathscr{S}$ is an $\mathscr{S}$-graph (called a *basic $\mathscr{S}$-graph*).

(ii) If F_1, F_2 are $\mathscr{S}$-graphs and if x is a procedure node of F_1 then $F_1(F_2 \text{ on } x)$ is an $\mathscr{S}$-graph (closure under composition) and $\text{seq}(F_1, F_2)$ is an $\mathscr{S}$-graph (closure under sequence).

$F =$ 

is non-sequential with components (i.e. maximal proper subflowgraphs)

$F_1 =$ 

$F_2 =$ 

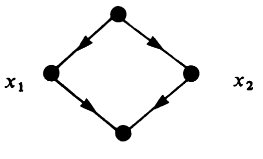
Note that $F(x_1 \text{ for } F_1, x_2 \text{ for } F_2) =$  $(= D_1)$

Figure 6

(iii) No flowgraph is an $\mathscr{S}$-graph unless it can be obtained by a finite number of applications of (i) and (ii).

The above definition is equivalent to the one in Ref. 4; if the flowgraph $P_2 \in \mathscr{S}$ then closure under sequence is a special case of closure under composition (Remark 2.4). The reason why we restrict the definition to families of irreducibles $\mathscr{S}$ is well documented in Ref. 4, where straightforward algorithms for determining whether an arbitrary flowgraph is an $\mathscr{S}$-graph are also given. This is a very elegant approach to structuredness since it is enough to specify the family $\mathscr{S}$ in order to characterise various notions of structuredness found in the literature. Thus for example if we take $\mathscr{S}^D = \{P_1, D_1, D_1', D_2, D_3\}$ the $\mathscr{S}^D$-graphs correspond to the usual interpretation of the 'D-structured' programs, although Prather has for simplicity omitted D_1' and D_3. More natural families to consider are the families $\mathscr{S}_n$ for each integer $n \geqslant 0$, where

$$\mathscr{S}_n = \{F : F \text{ is an irreducible with } \leqslant n \text{ predicate nodes}\}.$$

In the important case where we restrict ourselves to binary predicates, the set $\mathscr{S}_1$ is the same as $\mathscr{S}^D$ above with the addition of D_4, the so-called 'exit from middle' loop discussed at length in Ref. 9. Our derivation of $\mathscr{S}_1$ here seems to go some way to agreeing with Soloway, Bonar and Ehrlich, who have found no justification (on cognitive grounds) for the continued omission of D_4 in the definition of structured programs and more importantly of structured languages. In fact our concise definition of structuredness emphasises the ambiguity and (at times) total arbitrariness in the literature about 'D-structuredness'. The real power of our general approach will be seen when we wish to analyse a program's structure in order to metricate. For the purposes of metrication it is rather unsatisfactory if a given flowgraph cannot be metricated simply because it does not lie in a particular class of $\mathscr{S}$-graphs for some arbitrarily chosen family $\mathscr{S}$; we are interested in metrics which are defined on *all* flowgraphs. To this end we note that every flowgraph is an $\mathscr{S}_n$-graph for suitably large n, and moreover if

$$\mathscr{S}_\infty = \bigcup_{n=0}^{\infty} \mathscr{S}_n$$

then every flowgraph is an $\mathscr{S}_\infty$-graph, since $\mathscr{S}_\infty$ is just the family of all irreducibles. It was noted in Ref. 4 that every flowgraph has a uniquely defined 'irreducible hierarchy', i.e. explicit hierarchical description in terms of $\mathscr{S}_\infty$-graphs. If F is an $\mathscr{S}$-graph then each irreducible in this hierarchy is a basic $\mathscr{S}$-graph. In Ref. 8 Prather implicitly assumes a knowledge of this hierarchy for his $\mathscr{S}^D$-graphs in order to define metrics on $\mathscr{S}^D$-graphs. Similarly, to define metrics on all flowgraphs we will need to know in general exactly how to derive the irreducible hierarchy. Using the notation and definitions introduced in this section we are now able to provide a simple and elegant recursive procedure for this.

```
Procedure Tree ($F$)
    { returns a tree $T$ for a given flowgraph $F$.
      The nodes of $T$ are themselves flowgraphs }
begin
  if $F$ is irreducible then $T$ is a single node, $F$
  else begin
    Let $F_1, \ldots, F_k$ be the components of $F$;
    $F := F(x_1$ for $F_1, \ldots, x_k$ for $F_k)$ for identifiers
    $x_1, \ldots, x_k$;
    root ($T$): $= F$;
    subtrees of root ($T$) are Tree ($F_1$), ..., Tree ($F_k$)
    end {else}
end
```

It follows easily from earlier remarks that each node of Tree (F) is either a P_k (for some $k \geqslant 2$) or is an irreducible. Each occurrence of a P_k has k successor nodes (each of which is an irreducible, so that no ambiguity about 'nesting in sequences' occurs), meaning that at this point in the decomposition hierarchy there are k flowgraphs (one for each node) in sequence. Each occurrence of an irreducible F' means that each successor node was simultaneously nested on to F' at this point in the hierarchy. Figures 7 and 8 each show a flowgraph F and its irreducible hierarchy Tree (F), together with the derivation of the 'top-down' structure of F derived from Tree (F).

If $\mathscr{S}$ is the set of irreducibles appearing in the Tree (F), then $\mathscr{S}$ is the smallest family for which F is an $\mathscr{S}$-graph. Thus, for example, in Fig. 8 the family of irreducibles contains a 2-predicate node irreducible C_2 which means that F is not an $\mathscr{S}_1$-graph although it is an $\mathscr{S}_2$-graph. We also observe that the leaves of Tree (F) are always the set of irreducible subflowgraphs of F. An algorithm which continually decomposed irreducible subflowgraphs of F (as is given in Ref. 4) would effectively be reading the Tree (F) 'upwards'.

3. THE AXIOMS FOR METRICATION

The irreducible hierarchy is a definitive objective catalogue of the structure of a flowgraph. Notions such as size, type, frequency, depth/level of nesting of the irreducibles should all contribute to aspects of the structural complexity, and all of these notions are captured here. The irreducible flowgraphs are of course the 'atomic' structures in this analysis; it would be rash at this stage to assume that anything other than subjective 'complexity' metrics could be imposed on them (although we return to this issue in Section 4). However, we are interested in the minimal amount of subjectivity required for complete metrication (i.e. metrication of all flowgraphs, at least within a given class); in the light of the above analysis, once the complexity of the irreducibles is known, together with how complexity is affected by *sequence and nesting* (these represent the subjectivity), then the complexity of an arbitrary flowgraph should be uniquely determined with respect to these from the irreducible hierarchy.

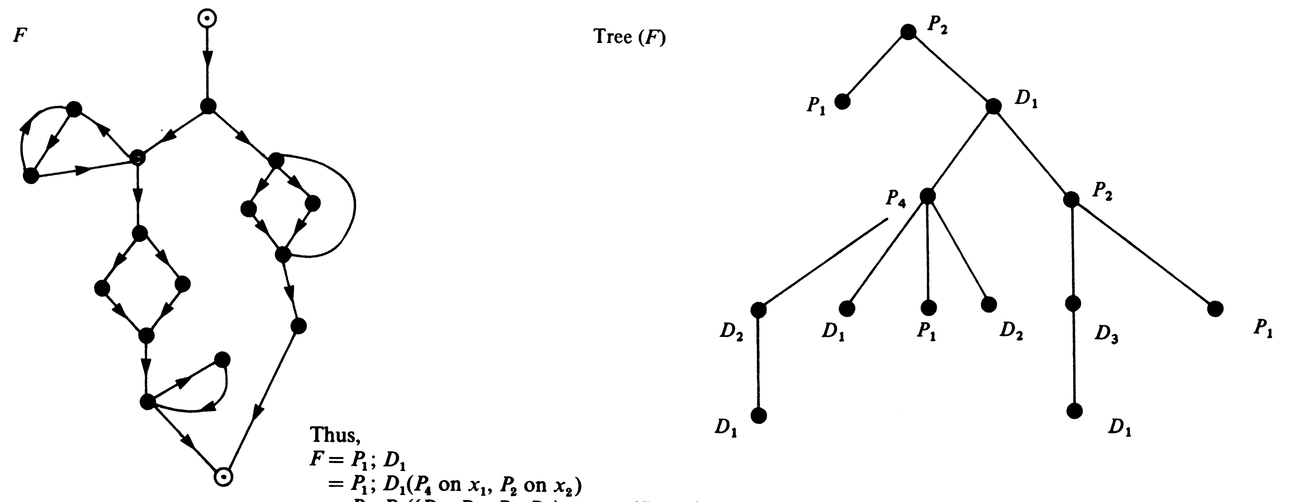
Formally, let us assume that

$$m : \mathscr{S} \to \mathbb{R}^+$$

is some predefined non-negative real-valued function on a family of irreducibles $\mathscr{S}$. We wish to *extend* the function m to the class of all $\mathscr{S}$-graphs. Our claim is that if m is indeed a 'complexity metric' then

(i) the complexity m of a sequential flowgraph should be uniquely determined by the complexities of the components, and

(ii) the complexity of a non-sequential flowgraph should be uniquely determined by the complexities of the components and the complexity of the flowgraph with all components decomposed.



Thus,

$F = P_1; D_1$
$= P_1; D_1(P_4 \text{ on } x_1, P_2 \text{ on } x_2)$
$= P_1; D_1((D_2; D_2; P_1; D_2) \text{ on } x_1; (D_3; P_1) \text{ on } x_2)$
$= P_1; D_1((D_2(D_2 \text{ on } x_3); D_1; P_1; D_2) \text{ on } x_1; (D_3(D_1 \text{ on } x_4); P_1) \text{ on } x_2)$

Figure 7

Specifically, we are laying down the following axioms for m

Axiom 1 (sequence axiom). For each $n = 1, 2, \ldots$ there is a function

$$g_n : \mathbb{R}^n \to \mathbb{R}^+$$

such that $m(\text{seq}(F_1, \ldots, F_n)) = g_n(m(F_1), \ldots, m(F_n))$ *and*

$$g_n \restriction \mathbb{R}^k = g_k \quad \text{for} \quad k = 1, \ldots, n$$

(meaning restriction to $\mathbb{R}^k$ considered as a subspace of $\mathbb{R}^n$, so e.g.

$$g_n(m(F_1), \ldots, m(F_k), 0, \ldots, 0) = g_k(m(F_1), \ldots, m(F_k)))$$

Axiom 2 (nesting axiom). For each irreducible F with procedure nodes $x_1, \ldots, x_n$ say, there is a function

$$h_F : \mathbb{R}^{n+1} \to \mathbb{R}^+$$

such that for any flowgraphs $F_1, \ldots, F_n$

$$m(F(F_1 \text{ on } x_1, \ldots, F_n \text{ on } x_n)) = h_F(m(F), m(F_1), \ldots, m(F_n)).$$

It now follows from the definition of $\mathcal{S}$-graphs that once $m(F)$ is defined for each $F \in \mathcal{S}$ and once the g_n and h_F functions of the axioms are known, then $m(F)$ is uniquely defined for each $\mathcal{S}$-graph F by considering the decomposition hierarchy of F with respect to $\mathcal{S}$. In particular, if $\mathcal{S} = \mathcal{S}_\infty$, then $m(F)$ is uniquely defined for any flowgraph F. Thus to define a metric m over $\mathcal{S}$ we will require only

definition of $m(F)$ for each F (M. 1)

definition of g_n for n 1, 2, ... (M. 2)

definition of h_F for each F. (M. 3)

Although Prather has not considered axioms like those above in his scheme, it is clear that they are implicitly assumed throughout his work.

A specific consequence of Axiom 2 is a restriction on the nature of h_F once $m(F)$ is defined, for if we take $F_i = P_1$ ($i = 1, \ldots, n$) then by remark 2.4(ii), we obtain:

Axiom 2(a). Each of the functions h_F satisfies

$$h_F(m(F), m(P_1), \ldots, m(P_1)) = m(F).$$

3.1. Examples of *g* and *h* functions

(i) A simple but uninteresting example of the g functions is obtained by taking $g_n = c$ for each n, where c is a non-negative constant. In the case of h functions, axiom 2(*a*) ensures a restriction on the type of constant functions allowable, but we could take $h_F = m(F)$ for each $F \in \mathcal{S}$.

(ii) Interesting examples of g functions are

$$g_n = \sum_{i=1}^{n} m(F_i) \quad \text{for each } n$$

$$g_n = \max(m(F_1), \ldots, m(F_n)) \quad \text{for each } n$$

However, $g_n = \min(m(F_1), \ldots, m(F_n))$ is invalid since, e.g. if $m(F_1) > 0$ then

$$g_2(m(F_1), 0) = 0 < m(F_1) = g_1(m(F_1))$$

which contradicts

$$g_s \restriction \mathbb{R}^1 = g_1$$

(iii) An interesting example for the h functions is

$$h_F(m(F), m(F_1), \ldots, m(F_n)) = m(F) . \max(m(F_1), \ldots, m(F_n))$$

Note however that

$$h_F = m(F) . \left(\sum_{i=1}^{n} m(F_i) \right)$$

is invalid generally because axiom 2(*a*) would then have to satisfy

$$m(F) = m(F) . \left(\sum_{i=1}^{n} m(P_1) \right) = n . m(F) . m(P_1),$$

i.e. $m(P_1) = 1/n$ for each n, which is impossible since $m(P_1)$ is constant. The 'next best' thing we can do here is to take

$$h_F = m(F) . \frac{1}{n} \left(\sum_{i=1}^{n} m(F_i) \right).$$

This *does* satisfy the axioms.

Axioms 1 and 2 are of course only part of the solution since they give no indication of how to define the g and h functions (nor of course the function m for the irreducibles). Certain aspects of complexity however suggest intuitive upper and lower bounds for the g and h functions; thus it is natural to suggest that a *general axiom scheme for m over $\mathcal{S}$* should have the additional axioms:

Axiom 3. For each n, there are functions $g_n^{\text{lower}}, g_n^{\text{upper}}: \mathbb{R}^n \to \mathbb{R}^+$ for which

$$g_n^{\text{lower}} \leqslant g_n \leqslant g_n^{\text{upper}}.$$

Axiom 4. For each $F \in \mathcal{S}$, there are functions $h_F^{\text{lower}}, h_F^{\text{upper}}: \mathbb{R}^{n+1} \to \mathbb{R}^+$ for which

$$h_F^{\text{lower}} \leqslant h_F \leqslant h_F^{\text{upper}}.$$

These two axioms say absolutely nothing more than axioms 1 and 2 unless we give *specific* functions for the upper and lower bounds, since we could assume

$$g_n^{\text{lower}} = h_F^{\text{lower}} = 0$$

$$g_n^{\text{upper}} = h_F^{\text{upper}} = \infty \text{ in each case.}$$

So a *specific axiom scheme for m with respect to $\mathcal{S}$* is a general axiom scheme in which axioms 3 and 4 have specific functions for the upper and lower bounds. Before considering non-trivial examples (like the one above) we have the following.

3.1. Theorem

Let $m_1, \ldots, m_t$ be metrics satisfying a specific axiom scheme with respect to $\mathcal{S}$. Then any positive weighted linear combination of these metrics also satisfies the specific axiom scheme.

Proof. A positive weighted linear combination is a metric

$$m = \sum_{i=1}^{t} a_i m_i,$$

where the a_is are constants for which

$$\sum_{i=1}^{t} a_i = 1$$

and each $a_i \geqslant 0$. Thus for each $F \in \mathcal{S}$ we define

$$m(F) = \sum_{i=1}^{t} a_i m_1(F)$$

and the g and h functions are similar sums of the respective g and h functions for the m_is. So, for example, if g_n^i is the g function for m_i then

$$g_n = \sum_{i=1}^{t} a_i g_n^i$$

is the g function for m. Now, for *each i*,

$$g_n^{\text{lower}} \leqslant g_n^i \leqslant g_n^{\text{upper}}$$

thus $\quad a_i g_n^{\text{lower}} \leqslant a_i g_n^i \leqslant a_i g_n^{\text{upper}},$

whence

$$g_n^{\text{lower}} = \left(\sum_{i=1}^{t} a_i\right) g_n^{\text{lower}} \leqslant \sum_{i=1}^{t} a_i g_n^i$$

$$= g_n \leqslant \left(\sum_{i=1}^{t} a_i\right) g_n^{\text{upper}} = g_n^{\text{upper}},$$

hence g_n satisfies Axiom 3. A similar argument is used for Axiom 4.

A weaker version of the theorem above is given by Prather in Ref. 8. The theorem will have important ramifications when we come to combine various types of metric in an attempt to achieve more general measures of complexity.

3.2. Examples of specific axiom schemes

(i) Let $\mathcal{S} = \{P_1, D_1, D_2\}$

For *Axiom 3* let $g_n^{\text{lower}} = \sum_{i=1}^{n} m(F_i)\, g_n^{\text{upper}} = \infty$

then the axiom reads

$$\sum_{i=1}^{t} m(F_i) \leqslant m(\text{seq}\,(F_1, \ldots, F_n)) < \infty$$

For *Axiom 4* we need only consider the irreducibles D_1, D_2 (these correspond respectively to **if** α **then** x_1 **else** x_2, **while** α **do** x)

(*a*) taking $h_{D_1}^{\text{lower}} = 1 + m(F_1) + m(F_2)$, $h_{D_1}^{\text{upper}} = 2(m(F_1) + m(F_2))$ yields

$$1 + m(F_1) + m(F_2) \leqslant m(D_1(F_1 \text{ on } x_1, F_2 \text{ on } x_2)) \leqslant 2(m(F_1) + m(F_2))$$

i.e.

$$1 + m(F_1) + m(F_2) \leqslant m(\textbf{if}\,\alpha\,\textbf{then}\,F_1\,\textbf{else}\,F_2) \leqslant 2(m(F_1) + m(F_2))$$

(*b*) taking $h_{D_2}^{\text{lower}} = 1 + m(F_1)$, $h_{D_2}^{\text{upper}} = 2m(F_1)$ yields

$$1 + m(F_1) \leqslant m(D_2(F_1 \text{ on } x_1)) \leqslant 2\, m(F_1)$$

i.e. $\quad 1 + m(F_1) \leqslant m(\textbf{while}\,\alpha\,\textbf{do}\,F_1) \leqslant 2\, m(F_1).$

The specific axiom scheme above is precisely the axiom scheme which Prather has proposed for '*D*-structured' flowgraphs, with the exception (as already noted) that our axioms 1 and 2 appear to have been only implicitly assumed throughout. There are a number of observations we wish to make regarding axiom 3, which is the most contentious here. First, no attempt has been made to set a finite upper bound on sequence complexity, which seems sad since surely this is the concept that is most easily dealt with at a cognitive level. Prather does point out that all 'reasonable' metrics ought to satisfy additivity of sequence (i.e. actually equal the lower bound) and hence implies that his axiom can become as strong as possible with an upper bound equal to the lower bound. We are of the opinion that this may be a reasonable assumption for metrics of global complexity, since this does correspond to intuitive notions. In certain cases, however, the summation may be a more reasonable upper bound than lower bound, particularly for measures reflecting aspects of local complexity, where a more realistic lower bound for sequence would be the maximum of the component complexities. In fact all examples of metrics considered in this paper will satisfy either

$$m(\text{seq}\,(F_1, \ldots, F_n)) = \sum_{i=1}^{n} m(F_i) \qquad (*)$$

or

$$m(\text{seq}\,(F_1, \ldots, F_n) = \max\,(m(F_1), \ldots, m(F_n)).$$

Prather has shown that an example of a metric which satisfies the axiom scheme (3.2.i) is McCabe's cyclomatic complexity,[7] which also meets the lower bound (*) for sequence. Halstead's metric of program volume[5] appears to satisfy the axioms in all but certain 'esoteric' cases, but in general the lower bound for sequence is not met, which would eliminate this metric for consideration within an axiom scheme where additivity of sequence was an *upper bound*. Prather's own metric μ satisfies the axiom scheme. In accordance with our earlier discussion this metric may be (equivalently) defined by:

(M. 1) $\mu(P_1) = 1$, $\mu(D_1) = \mu(D_2) = 2$ (*Note*: P_1 corresponds to Prather's 'simple statement')

(M. 2) $\mu(\text{seq}(F_1, \ldots, F_n)) = \sum_{i=1}^{n} \mu(F_i)$ for each n (g_n function)

(M. 3) (i) $\mu(D_1(F_1 \text{ on } x_1, F_2 \text{ on } x_2)) = \mu(D_1) \max (\mu(F_1), \mu(F_2))$ (h_{D_1} function)

(ii) $\mu(D_2(F_1 \text{ on } x_i)) = \mu(D_2)\mu(F_1)$ (h_{D_2} function)

On the basis of this definition it is routine to check that μ satisfies the specific axiom scheme 3.2.i. Although it is only defined for a small (albeit significant) family $\mathscr{S}$, the metric μ appears to be a far more satisfactory metric than McCabe's for reflecting structural complexity. In particular it reflects the multiplicative effect that nesting intuitively contributes towards complexity. As with any of the subjective ingredients of a metric (i.e. the statements M. 1, M. 2, M. 3) there are grounds for questioning the legitimacy in certain circumstances. For example, the values $\mu(D_1)$, $\mu(D_2)$ ($=2$) appear relatively low in relation to $\mu(P_1) = 1$. In particular, we find that $\mu(P_n) = n$ for each n, suggesting that $\mu(D_1)$, $\mu(D_2)$ are less complex than any sequence of $\geqslant 3$ simple statements. In general, for any metric m the most effective way to assign a value to $m(F)$ for an irreducible F once $m(P_1)$ is known will be to 'assess' how many sequential statements are needed in order to equal the perceived complexity of F (i.e. to assess what value of n is P_n closest in complexity to F). However, this problem is perhaps more suited to cognitive psychologists than to computer scientists, and it is doubtful if there will ever be universal agreement, even if the aspect of complexity in question is clearly defined.

The main drawback of μ, and indeed Prather's axioms, is the limited class of flowgraphs on which they are defined. This problem is now easily overcome with our theory. We may extend Prather's axioms and metric μ to the most general of flowgraphs, namely the $\mathscr{S}_\infty$-graphs.

3.2.ii. *A specific axiom scheme for* $\mathscr{S}_\infty$

By *axiom 3*

$$\sum_{i=1}^{n} m(F_i) \leqslant m(\text{seq}(F_1, \ldots, F_n)) < \infty$$

By *axiom 4* for each $F \in \mathscr{S}_\infty$, with procedure nodes $x_1, \ldots, x_n$

$$\max(m(F), m(F_1), \ldots, m(F_n)) \leqslant m(F(F_1 \text{ on } x_1, \ldots, F_n \text{ on } x_n)) \leqslant m(F)\left(\sum_{i=1}^{n} m(F_i)\right)$$

i.e.

$$h_F^{\text{lower}} = \max(m(F), m(F_1), \ldots, m(F_n))$$
$$h_F^{\text{upper}} = m(F)\left(\sum_{i=1}^{n} m(F_i)\right).$$

As already discussed, variations on this axiom scheme will be considered in which axiom 3 is changed. In particular we consider:

3.2.iii. *A specific axiom scheme for* $\mathscr{S}_\infty$

For *axiom 3* $\max(m(F_1), \ldots, m(F_n)) \leqslant m(\text{seq}(F_1, \ldots, F_n)) < \infty$

For *axiom 4* as for 3.2.ii

3.2.iv. *A specific axiom scheme for* $\mathscr{S}_\infty$

For *axiom 3* $\max(m(F_1), \ldots, m(F_n)) \leqslant m(\text{seq}(F_1, \ldots, F_n)) \leqslant \sum_{i=1}^{n} m(F_i)$

For *axiom 4* as for 3.2.ii

Scheme 3.2.ii is clearly an extension of Prather's scheme. Scheme 3.2.iii is weaker in the sense that it implies scheme 3.2.ii but has a more realistic lower bound for axiom 3. Scheme 3.2.iv retains this lower bound but imposes the additive upper bound, which we feel is more consistent with the upper bound of axiom 4.

A metric μ' which extends μ to the class of $\mathscr{S}_\infty$-graphs is now given by:

(M. 1) For each $F \in \mathscr{S}_\infty$, $\mu'(F) = \pi + 1$

where π = number of predicate nodes of F

(M. 2) $\mu'(\text{seq}(F_1, \ldots, F_n)) = \sum_{i=1}^{n} \mu'(F_i)$ for each $n(g_n)$

(M. 3) $\mu'(F(F_i \text{ on } x_i, \ldots, F_n \text{ on } x_n)) = \mu'(F) \max (\mu'(F_1), \ldots, \mu'(F_n))$ for each $F \in \mathscr{S}_\infty$ (h_F).

It is routine to check that μ' is a metric satisfying the axiom scheme 3.2.ii (and also 3.2.iii and 3.2.iv), which *extends* μ. We do note how axiom 2.a is established:

$$\begin{aligned} &\mu'(F(P_1 \text{ on } x_1, \ldots, P_1 \text{ on } x_n)) \\ &= \mu'(F)\max(\mu'(P_1), \ldots, \mu'(P_1)) \\ &= \mu'(F)\,1 \\ &= \mu'(F) \text{ as required.} \end{aligned}$$

3.4. Example

For the flowgraph F in Fig. 8, $\mu'(F) = 8$.

The values of $\mu'(F)$ for each $F \in \mathscr{S}_\infty$ (defined in M. 1) are subject to the same criticisms as we noted for the subset already defined for μ. Other interesting metrics which attempt to overcome some of these problems may be constructed by redefining M. 1 (and leaving M 2, M. 3 changed). For example we could define the metrics μ'', λ based on

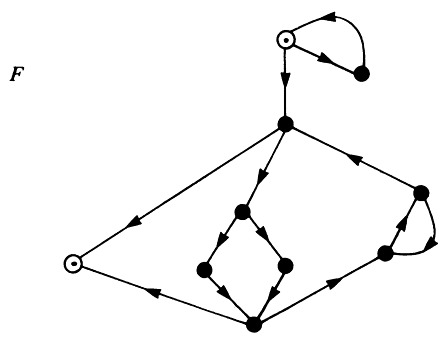
(*a*) M. 1 (for μ'') $\mu''(F) = 2^\pi$ for each $F \in \mathscr{S}_\infty$

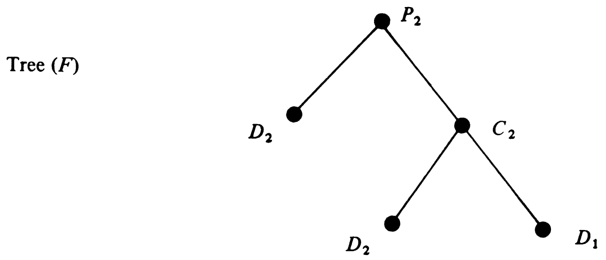
(*b*) M. 1 (for λ) $\lambda(F) = \pi^2 + 1$ for each $F \in \mathscr{S}_\infty$

(where in each case π = number of predicates in F).

Both these metrics are, like μ', extensions of μ giving greater (but differing) emphasis to 'larger' irreducibles. For the flowgraph F of Fig. 8 we have $\mu''(F) = 10$, and $\lambda(F) = 12$.

Each of the metrics considered so far associates equal complexity to irreducibles with the same number of





Thus,
$F = D_2; (C_2(D_1 \text{ on } x_1, D_2 \text{ on } x_2))$
Hence
$$\mu'(F) = \mu'(D_2)+\mu'(C_2(D_1 \text{ on } x_1, D_2 \text{ on } x_2))$$
$$= 2+\mu'(C_2).(\max(\mu'(D_1), \mu'(D_2))$$
$$= 2+3 (\max (2, 2)) = 8$$

Figure 8

predicate nodes. While this enables us to give elegant and simple statements for part M. 1 of the definition of such metrics, this is in general an over-simplification of complexity as perceived. The only definitive solution is to consider *each* irreducible F on its own merit and to subject it to exhaustive tests, like those done for the D-structures in Ref. 9; even then the software community may not be satisfied, since there will be the inevitable trade-off between metrics being more 'finely tuned' and being more difficult to calculate in practice.

Whatever their respective merits, metrics like μ', μ'', λ and other similar ones which the reader may easily construct now for himself (and which are soon to be the subject of large-scale industrial tests), appear to be reasonable candidates for measures of *global* complexity. Frequently however we are interested, for purposes of quality control and the like, in 'worst-case' complexity – i.e. some kind of measure (possibly reflecting *localised* complexity) which, if exceeded, lead to the rejection of the software on the grounds that the measure is sufficiently high for the whole software to be considered unacceptable. A useful example of such a measure is the so-called 'structural complexity metric' κ analysed (together with its linguistic properties which are beyond the bounds of this paper) in Ref. 3. Informally $\kappa(F)$ was defined for any flowgraph F as being the smallest integer n for which F is an $\mathcal{S}_n$-graph, or equivalently $\kappa(F)$ is the 'size' of the largest irreducible in the decomposition hierarchy of F (where size = number of predicate nodes). Thus F is 'D-structured' if and only if $\kappa(F) = 1$; a value $\kappa(F) > 1$ would merit rejection of F if our criteria for acceptance of software were that it had to be 'D-structured'. A less harsh procedure for quality control would isolate those components of F which record maximal complexity and seek to restructure these.

There are a number of objective comments that we can make about κ in the light of the axiomatic approach. First we note that κ may be equivalently defined by:

(M. 1) For each $F \in \mathcal{S}_\infty$, $\kappa(F) = \pi$ (number of predicate nodes)

(M. 2) $\kappa(\text{seq}(F_1, \ldots, F_n)) = \max(\kappa(F_1), \ldots, \kappa(F_n))$

(M. 3) $\kappa(F(F_1 \text{ on } x_i, \ldots, F_n \text{ on } x_n)) = \max(\kappa(F), \kappa(F_1), \ldots, \kappa(F_m))$.

From this definition of κ it is clear that κ satisfies the specific axiom schemes 3.2.iii and 3.2.iv. In fact the g_n and h_F functions actually meet the lower bounds.

In the example of Figure 7, $\kappa(F) = 2$, indicating that a 2-predicate node irreducible (in this case a single occurrence of C_2 in the hierarchy) is the obstruction to 'D-structuredness'.

It is worth emphasising at this point another merit of the generalised approach to structuredness; many people who believe in the notion of structured programming are of the opinion that the D-structures are too limited a class and that some constructs (notably a multi-exit loop for control environments) ought to be added. In order to accommodate this our definition of structuredness does not change at all – only the family $\mathcal{S}$ needs to be extended. Thus to allow multi-exit loops we simply add, say to $\mathcal{S}_1$ the set of appropriate flowgraphs (in this case those whose CGK-graph is an n-cycle with n exit arcs – see Ref. 4). An analogous metric κ' to κ may now be defined by taking $\kappa'(F) = 1$ for each F in our chosen family $\mathcal{S}$, and $\kappa'(F) = \pi$ as before for other irreducibles F. The rest of the definition of κ' remains as for κ. The resulting metric κ' which again satisfies axiom schemes 3.2.iii and 3.2.iv is such that for any flowgraph F, $\kappa'(F) = 1$ if and only if F is an $\mathcal{S}$-graph. A high value of $\kappa'(F)$ would suggest a genuinely high degree of tangled logic contained in at least one component of F.

Another metric which is in a similar category to κ, κ' is the metric α, which records the maximum depth of nesting. Specifically α may be defined by:

(M. 1) For each $F \in \mathcal{S}_\infty$, $\alpha(F) = 1$, except $\alpha(P_1) = 0$

(M. 2) $\alpha(\text{seq}(F_1, \ldots, F_n)) = \max(\alpha(F_1), \ldots, \alpha(F_n))$ (g_n)

(M. 3) $\alpha(F(F_1 \text{ on } x_1, \ldots, F_n \text{ on } x_n)) = 1 + \max(\alpha(F_1), \ldots, \alpha(F_n))$ (h_F for irreducibles F).

Again, it is routine to check that α satisfies the axiom schemes 3.2.iii and 3.2.iv.

4. FUTURE DEVELOPMENTS

In the light of the many different types of metrics considered in this paper as measures of certain aspects of structural complexity, it is impossible to exaggerate the importance of theorem 3.1. Depending on which aspects of complexity are deemed most important, suitable metrics may eventually be derived by attaching appropriate weights to any number of metrics (satisfying the same specific axiom scheme) which are considered to be significant. Ultimately the goal is towards a 'general complexity' metric; the closer we get to this – and hence the closer we get towards an understanding of what genuinely contributes complexity – the more restrictive we shall be able to make the upper and lower bounds of our axiom schemes.

In the meantime we pose some specific problems which are worthy of urgent attention.

(1) The metric κ provides an example of a metric which meets the lower bound functions in axiom schemes 3.2.iii and 3.4.iv, most notably the lower bounds of axiom 4. However, it has been noted (in 3.1.iii) that no metric can possibly meet all the upper bounds of axiom 4 in these schemes. This suggests a possible 'relaxation' of the upper bounds in axiom 4 and hence more restrictive axiom schemes. What kind of improved upper bounds should we be looking for?

(2) We have assumed throughout that the axioms represent the objective attributes for metrics and the M. 1, M. 2 and M. 3 defined for a given metric represent the wholly subjective attributes. In fact certain implicit assumptions about the latter may well be incorporated as *extra* axioms and thus lessen the apparent subjectivity. In particular consider M. 1, i.e. the definition of $m(F)$ for the irreducibles F. In each example of a metric seen above we have $m(P_1) = 1$, suggesting quite reasonably that the flowgraph P_1 corresponding to a simple statement should be the basis for metrication in all cases. This represents the best candidate for an extra axiom (in the absence of linguistic considerations). In general, as we discover more about the properties of other irreducible F, the possibility of incorporating this knowledge into the axiom scheme will increase. If, for example, it can be established that an irreducible F is more 'complex' than another F' (by whatever means are available) then any metric m which is expected to reflect this complexity ought to satisfy $m(F) > m(F')$ as an axiom. This suggests that the most urgent need for research in this area lies in the comparative evaluation of irreducibles' complexity. We already have some inroads on this problem; our more restrictive definition of subflowgraph in this paper (compared to Ref. 4) means that the class of irreducibles here is larger than that in Ref. 4. The 'extra' irreducibles here can be 'unfolded' (see Ref. 4) in terms of smaller flowgraphs (namely those which had multi-entry points). The operation of unfolding should eventually be treated by axioms similar to those for sequence and nesting.

(3) Suppose F, F' are complexity metrics with $m(F) < m(F')$. Certain intuitive notions of complexity suggest that a complex structure nested inside a simple structure ought to be less complex overall than the same simple structure nested inside the complex structure. In the former the real complexity is localised (and hence can be 'controlled' separately), while in the latter the real complexity is global, and hence more difficult to control. This notion suggests an additional axiom, namely

$$m(F) < m(F') \Rightarrow m(F(F' \text{ on } x)) < m(F' \text{ on } y)).$$

Unfortunately, the metrics considered so far do *not* satisfy this property since in each case we have

$$m(F(F' \text{ on } x)) = m(F'(F \text{ on } y)).$$

To define metrics which do have this property we shall have to give greater 'weight' to the flowgraph in which the nesting takes place. To achieve this, the h_F functions defined in M. 2 could be, for example

$$m(F(F_1 \text{ on } x_1, \ldots, F_n \text{ on } x_n) = m(F)^b \max(m(F_1), \ldots, m(F_n)), \quad \text{where} \quad b > 1.$$

For such metrics we may require higher upper bounds in axiom 4 of any specific axiom scheme (than in the examples earlier).

REFERENCES

1. B. Curtis, S. B. Sheppard, P. Milliman, M. A. Borst and L. T. Love, Measuring the psychological complexity of software maintenance tasks with the Halstead and McCabe metrics. *IEEE Transactions, Software Engineering* vol. SE-5 (2), 96–104 (1979).
2. W. H. Evangelist, Software complexity metric sensitivity to program structuring rules. *JSS* **3**, 231–243 (1983).
3. N. E. Fenton, The structural complexity of flowgraphs. In *Proceedings of the 5th International Conference on the Theory and Applications of Graphs*. Wiley, Chichester (1985).
4. N. E. Fenton, R. W. Whitty and A. A. Kaposi, A generalised mathematical theory of structured programming. *Theoretical Computing Science* **38** (1985).
5. M. H. Halstead, *Elements of Software Science*. Elsevier North-Holland, Amsterdam (1975).
6. B. A. Kitchenham, Measurements of programming complexity. *ICL Technical Journal* 298–316 (1981).
7. T. J. McCabe, A complexity measure. *IEEE Transactions*, SE-2, 308–320 (1976).
8. R. E. Prather. An axiomatic theory of software complexity measure. *The Computer Journal* **27**, 340–347 (1984).
9. E. Soloway, J. Bonar and K. Ehrlich, Cognitive strategies and looping constructes: an empirical survey. *Communications, ACM* **26** (11), 853–860 (1983).
10. R. W. Whitty, N. E. Fenton and A. A. Kaposi, A rigorous approach to structural analysis and metrication of software. *IEE Software and Microsystems* **4** (1), 2–16 (1984).
11. R. W. Whitty, N. E. Fenton and A. A. Kaposi, Structured programming: a tutorial guide. *IEE Software and Microsystems* **3** (3), 54–65 (1984).
12. M. R. Woodward, M. A. Hennel and D. Hedley, A measure of control flow complexity in program text. *IEEE Transactions* SE-5 **1**, 45–50 (1979).
13. R. E. Prather and S. G. Giuleri, Decomposition of flowchart schemata. *The Computer Journal* **24** (3), 258–262