# A Philosophy for Software Measurement*

Albert L. Baker
Department of Computer Science
Iowa State University
Ames, Iowa 50011

James M. Bieman
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523

Norman Fenton†
Centre for Software Reliability
The City University
Northampton Square
London EC1V 0HB
United Kingdom

David A. Gustafson
Department of Computing and Information Sciences
Kansas State University
Manhattan, Kansas 66506

Austin Melton‡
Department of Computing and Information Sciences
Kansas State University
Manhattan, Kansas 66506

Robin Whitty
Department of Electrical and Electronic Engineering
Polytechnic of the South Bank
Borough Road
London SE1 0AA
United Kingdom

April 4, 1990

to appear in *The Journal of Systems and Software*

**Abstract**

We as a group – called the Grubstake Group – are convinced that software measures are essential for "controlling" software. Thus, we are dedicated to producing an environment in which software measures can be confidently used by software managers and programmers. However, we are also convinced that such an environment can only be created if there exists a formal and rigorous foundation for software measurement. This foundation will not have to be understood by the users of the software measures, but it will have to be understood by those who define, validate, and provide tool support for the measures. It is this foundation which we are introducing in this paper.

*Research is supported in part by NATO Collaborative Research Grant 034/88.
†Research is supported in part by ESPRIT project PDCS and by British Telecom.
‡Research is supported in part by ONR Grant N00014-88-K-0455.

# 1 Basic Position

Since Maurice Halstead developed his software science metrics, software researchers and practitioners have sought meaningful software measures. Researchers seek quantitative measures of software "quality" and "complexity", and practitioners see software measures as tools to control the growing cost of software development and maintenance. However, the research literature contains many negative criticisms of software measures research. The field of software measurement has been criticized for poor empirical methodology and for a lack of theoretical foundations.

In spite of the criticism, we believe that software measurement is important, and we further believe that software measurement can become a trusted and well-respected branch of software engineering. We want to be able to take meaningful measurements of software documents and the software production process. We also want to be able to use software measures to make accurate predictions. Unfortunately, software measurement research is often suspect due to a lack of rigor and/or unjustified claims. For research results to be meaningful, software measurement must be well grounded in theory. Further, empirical results must be obtained through well designed experimental work.

Our philosophy is to use the science of measurement theory [5] for the foundation of software measurement. Measurement theory, which is the basis for measurement in the physical sciences, provides a framework for numerically characterizing intuitive properties or attributes of objects and events. Applying the basic criteria of measurement theory to software measures requires the identification and/or definition of

- attributes of software products and processes. These attributes need to be aspects of software that have both intuitive and well-understood meanings. For example, the attribute of length for the product of source code satisfies this whereas coupling of software designs or source code has a number of different interpretations. However, we believe that a dialogue among software professionals could lead to a understanding of the term "coupling".

- formal models or abstractions which capture the attributes. For example, one possible model of source code for capturing the attribute of length is the representation of the source code in binary form via the 8-bit ASCII code.

- important relationships and orderings which exist between the objects (being modeled) and which are determined by the attributes of the models. For example, if we understand what length means and we have a model of length, we can compare two programs in terms of their relative length. Such comparisons impose a "length order" on the software documents.

- mappings from the models to number systems which preserve the order relationships. This idea is the basis of the representational theory of measurement. For example, any measure of length must not contradict the length ordering that our model and our intuitive understanding of length imposes.

If all the above criteria are satisfied, then the resulting mapping will be called a software measure. An arbitrary mapping which does not necessarily satisfy the first three requirements is a metric. This distinction between a software measure and metric is consistent with the interpretation provided in [6]. We believe that much of the criticism in the professional literature stems from metrics being presented as measures.

We would also like to be able to use measurement to make valid predictions. For example, measures of attributes of a software specification could be used to make predictions about attributes of an implementation. But, in order to predict, we need

- two or more measures of attributes. For example, one measure may be of a specification attribute and the other may be of an implementation attribute.
- a theory that relates the measures. One cannot randomly look for correlations between two measures. One must have an **a priori** theory or hypothesis to establish causality.
- an empirical demonstration that the theory holds.

Research on both developing meaningful measures and establishing relationships between measures is necessary. In order to use software measurements to make predictions, the research emphasis needs to be on establishing theories relating different measures. We further believe that a measure need not be a part of a prediction system to be valid and useful.

## 2 Validation

Numerous software measures are described in the literature. These are designed to measure a wide range of attributes, and they naturally vary greatly in definition and in use. Unfortunately, for most of these irreconcilable measures, we find research reports claiming that they measure or predict similar attributes such as cost, size, and complexity. This state of affairs is commonly attributed to a general lack of **validation** of software measures. While accepting this reason, we propose more fundamentally that there is a lack of understanding of the meaning of validation of software measures.

However, before we discuss validation, we need to be clear about what we are validating. Literature references to software measures actually refer to two separate concepts:

- **Measures** which are defined on certain objects and characterize numerically some specific attribute of these objects. (This is how we have defined software measures.)
- **Prediction systems** involving a mathematical model and prediction procedures for it.

The software metrics community has generally not differentiated between these two concepts resulting in confusion surrounding the notions of, and obligations for, validation. The two concepts require different types of validation. In the case of measures we turn to measurement theory for the notion of validation.

**Definition 2.1** *Validation* of a software measure is the process of ensuring that the measure is a proper numerical characterization of the claimed attribute.

This type of validation is central in our use of measurement theory. Practitioners may prefer to regard this as ensuring the well-definedness and consistency of the measure. To stress this where necessary we may also refer to it as **internal validation** since it may require consideration of the underlying models used to capture the objects and attributes.

For prediction systems we have:

**Definition 2.2** *Validation* of a prediction system is the usual empirical process of establishing the accuracy of the prediction system in a given environment by empirical means, i.e., by comparing model performance with known data points in a given environment.

So where does the confusion arise?

**It arises out of a basic (and poorly articulated) misconception that a software measure must always be part of a prediction system.**

The misconception is normally presented in something like the following manner:

A software measure is only **valid** if it can be shown to be an accurate predictor of some software attribute of general interest like cost or reliability.

This misconception is so damaging for scientific approaches to software measurement and validation that it is worth dismantling it by formal arguments. (Before we list these arguments, we should point out that some practitioners use "valid" to mean of worth or reasonable and not "scientifically valid". This difference should be kept in mind when reading argument 2 below.)

1. This view of validation is ill-defined since it is not known which rigorously measurable software attributes are directly related to the general interest attributes such as cost and reliability.
2. This view of validation contradicts the very meaning of measurement.
3. A scientific approach to the prediction problem requires that any predictive capability of a measure be stated as a hypothesis. In this case validation is clearly understood, since this is the case of the proposal of a prediction system.

Let us briefly consider an important ramification of this measure-predictor misconception:

Suppose that we have some good measures of internal product attributes, like size, structuredness, modularity, functionality, coupling, and cohension. Then it is apparently not enough that these measures accurately characterize the stated attributes because these are not considered to be of "general" interest. Since there is generally no specific hypothesis about the predictive capabilities of such measures, they are shown to be "valid" by correlation against any "interesting" measures which happen to be available as data. For example a measure of coupling might be claimed to be valid or invalid on the basis of a comparison with known **development costs** if the latter is the only "data" available. This would be done even though no claims were ever made about a relationship between coupling and development costs!

It is conceivable that a measure could be shown to be valid in the sense of its being a component of a valid prediction system even though no hypothesis existed. For this to occur the "data" which happens to be available would have to be shown to be consistently related via a formula determined initially by regression analysis. If such validation does occur, let us call it **external validation** of the measure to distinguish it from the (internal) validation which should initially take place to show that it actually measures some attribute.

**Definition 2.3** *External validation* of a measure m is the process of establishing a consistent relationship between m and some available empirical data purporting to measure some useful attribute.

Given our poor understanding of the relationships between various software products and processes, external validation seems highly unreliable. **And yet we are expected to accept that this as the major approach to validation!**

The reason we are interested in measuring internal attributes like size, structuredness, modularity, control flow complexity, data flow complexity, cohesion, and coupling, is because we believe that not only are these important concepts in their own right but also because they will necessarily play a role in many types of prediction systems. Indeed in the case of size, we already note that it is a component of almost all cost and productivity models and prediction systems; at the very least we should already have ensured that the proposed measures of size capture our intuitive understanding of size!

# 3 Developing Structural Measures

We usually define an "attribute" or an "aspect" of software documents when we use a particular **abstraction** of the software document. For example, the number of linearly independent paths in a program (also known as the cyclomatic number) is actually defined for the flowgraph abstraction of the program. A methodology for defining a structural measure is: first define an **abstraction** of the document, then define an order on the abstraction, and then define an order-preserving map from the abstraction to the real numbers (or other appropriate number system).

A consideration of different types of orders on the sets of abstractions leads to a hierarchy of structural measures.

**Definition 3.1** Let $D$ be a set of similar documents; let $A$ be a set of abstractions; let *abs*: $D \longrightarrow A$ be the function which maps each object to its abstraction; $\leq_A$ be a partial order on $A$; and let $m^\star : (A, \leq_A) \longrightarrow (\Re, \leq_\Re)$ be an order preserving map. Then $m(D) = m^\star(abs(D))$ is a *well-founded structural measure.*

We can define a subset of well-founded structural measures by restricting the nature of the orders on abstractions. Consider a partial order $\leq$ which formalizes a notion of **containment**. For example in the set of flowgraphs, we could have $G \leq G'$ exactly when flowgraph $G$ is "contained" in a flowgraph $G'$. For an exact definition of $\leq$ on the set of flowgraphs, see [1]. This notion of a containment-based partial order can be used to define a subset of the set well-founded structural measures.

**Definition 3.2** Let $D$ be a set of similar documents; let $A$ be a set of abstractions; let *abs*: $D \longrightarrow A$ be the abstraction map; let $\leq_A$ be a containment-based partial order on $A$; and let $m^\star : (A, \leq_A) \longrightarrow (\Re, \leq_\Re)$ be an order preserving mapping. Then $m(D) = m^\star(abs(D))$ is a *containment-based structural measure.*

The definitions of a well-founded structural measure and a containment-based structural measure suggest a methodology for software measures research. First, when defining a measure one needs to specify precisely the documents and the attribute to be measured. Then an abstraction or model which captures the attribute and an order which respects the attribute must be given. Finally an order preserving map from the abstractions or model to a number system is defined. The order-preservation means that the map is indeed capturing the attribute.

# 4 Immediate and Realistic Uses for Structural Software Measures

For a valid software measure to be useful it is not necessary that it be part of a prediction system or that it be a measure of "understandability" or "psychological complexity". Successful measures of structural properties can be developed when important structural properties can be unambiguously identified, modeled, and understood.

The software attribute that we examine is the "difficulty" of applying a particular testing strategy to software. A measurable component of this "difficulty" attribute is the number of test cases needed. A measure that accurately estimates the number of required tests obviously satisfies all reasonable intuition concerning the difficulty of using a testing strategy.

Structural measures are most applicable to structural testing strategies. Structural testing prescribes that particular sets of program paths with certain structural properties be tested. For example, the criteria proposed by Rapps and Weyuker [4] require testing specific sets of paths

that follow the flow of data from expressions through assignments to other expressions. The **all-du-paths** criterion is the strongest of these criteria. Unfortunately, it may potentially take an enormous number of tests to satisfy the criterion. In the worst case it takes $2^t$ number of tests, where $t$ is the number of branches [7].

An alternative to a worst case analysis is to use structural measures to estimate the number of test cases actually required to satisfy a criterion on specific programs. Such measures can simply count the number of complete paths through a program necessary to meet a criterion. In defining such a measure, we are working with the set of coded programs. The attribute which we want to measure is the cardinality of a minimal set of test data satisfying test strategy requirements. Our abstraction is the set of flowgraphs annotated with data flow information.

Using such a measurement tool, Bieman and Schultz conducted an investigation to determine how many test cases are actually needed to satisfy the all-du-paths testing criterion [2, 3]. The tools developed by Bieman and Schultz identify minimal sets of complete program paths that satisfy data flow criteria. The cardinality of these minimal sets measures the estimated number of test cases needed to satisfy the criteria. The measures were used in an empirical study with results that indicate that the all-du-paths criterion is much more practical than previous analytical results suggest. In their study of a commercial software system, Bieman and Schultz found that the all-du-paths criterion can usually be satisfied by testing fewer than ten complete paths, and units requiring an exponential number of tests are rare.

This example shows that researchers and practitioners can gain valuable insights when they focus on the direct implication of a structural measure. A measure that determines the size of a minimal set of paths necessary to satisfy a structural criterion is quite useful. The number of test cases needed for a particular testing strategy is clearly a component of the difficulty of using the strategy. Such a measure can be used to determine the practicability of the criterion. We can also use such a measure to estimate the required number of test cases and identify hard to test program units. Such a measure need not predict the elusive "understandability" property to be useful.

## 5 Conclusions

Software measurement is critical and necessary in order to provide a scientific basis for software engineering. Meaningful software measurements must be well-grounded and measurement theory is the natural foundation. We, the Grubstake Group, have introduced the application of measurement theory to software measures, described implications of the theory to validation and to the development of structural measures, and have shown that useful measures can be developed under the measurement theory framework.

## References


[1] A. Baker, J. Bieman, D. Gustafson, and A. Melton. Modeling and measuring the software development process. *Proc. 20th Hawaii International Conference on Systems Sciences (HICSS-20)*, II:23–30, January 1987.

[2] J. Bieman and J. Schultz. An empirical evaluation of the all-du-paths testing criterion. submitted to *Software Practice & Experience*, 1989.

[3] J. Bieman and J. Schultz. Estimating the number of test cases required to satisfy the all-du-paths testing criterion. *Proc. Software Testing, Analysis and Verification Symposium (TAV3–SIGSOFT89)*, pages 179–186, December 1989.

[4] S. Rapps and E. J. Weyuker. Selecting software test data using data flow information. *IEEE Trans. Software Engineering*, SE-11(4):367–375, April 1985.

[5] F.S. Roberts. *Measurement Theory with Applications to Decision Making, Utility, and the Social Sciences*. Addison Wesley, 1979.

[6] V.Y. Shen S.D. Conte and H.E. Dunsmore. *Software Engineering Metrics and Models*. Benjamin Cummins Publishing, Inc., 1986.

[7] E. J. Weyuker. The complexity of data flow criteria for test data selection. *Information Processing Letters*, 19:103–109, August 1984.

# 6 Biographies

## 6.1 Albert L.Baker

Dr. Albert L. Baker is currently an Associate Professor in the Department of Computer Science at Iowa State University. His software engineering research intertests include specification languages, software measures, and programming methodologies. His work in software measures has for some time focused on providing a more rigorous basis for definitions and analyses of software measures. His work in specification languages and methodologies is focused on bringing increased precision to some of the less formal specification techniques currently in use in production environments, e.g., structured analysis (SA) techniques. This research on increasing the precision of SA specifications has been supported in part by Rockwell International Corporation, Cedar Rapids, Iowa.

Dr. Baker also conducts research and development in natural language text analysis systems. The goal of this research is to produce practical systems that can be used to more systematically analyze large volumes of textual data, e.g., as might be generated in a large population survey consisting of open-ended questions requiring verbatim responses.

## 6.2 James M. Bieman

Dr. James M. Bieman is an Associate Professor of Computer Science at Colorado State University, Fort Collins, Colorado. He was previously an Assistant Professor at Iowa State University. Dr. Bieman's research is focused on the structural analysis of software, software measures, software testing, and executable specification languages. He has developed software measurement tools to estimate the required number of test cases necessary to meet data flow testing criteria. Results of a case study of a commercial software system suggest that the strongest of the data flow criteria, the all-du-paths criterion, is much more practical than the theoretical results indicate. His research on executable software specifications demonstrate that specification assertions can be effectively expressed within executable type expressions. Dr. Bieman (with J. Leszczyłowski) designed the PROSPER executable specification language.

## 6.3 Norman E. Fenton

Dr. Norman E. Fenton, PhD, C.Eng, MIEE, AFIMA, is Reader in Software Engineering at the Centre for Software Reliability, City University, London. His previous positions include being Director of the Centre for Systems and Software Engineering and Reader in Information Technology at South Bank Polytechnic and Research Fellow in Mathematics at Oxford University. Dr. Fenton has published widely on various topics in software engineering and discrete mathematics, with recent emphasis on theories of structured programming and software metrics. He is currently concerned with evaluation of both software products and methods for software production and in the formal foundations of software measurement generally. Dr. Fenton is involved in several major industrial collaborative projects as researcher or consultant. These include various ESPRIT projects in software engineering, metrics, and software certification. Dr. Fenton has been a member of numerous conference organizing committees and software standards bodies committees.

## 6.4 David A. Gustafson

Dr. David A. Gustafson received the B.Math from University of Minnesota, the BS (Meteorology) from the University of Utah, and the MS and PhD in Computer Science from the University of Wisconsin - Madison.

He is an Associate Professor in the Computing and Information Sciences Department at Kansas State University, Manhattan, Kansas. His research interests include the theory and validation of software measures, software reliability, software testing methods, formal notations for diagrams, and notations for describing the software development process.

Dr. Gustafson is a member of IEEE, ACM, MAA, Sigma Xi, UPE and Tau Beta Pi.

### 6.5 Austin C. Melton

Dr. Austin C. Melton is an Associate Professor in the Department of Computing and Information Sciences at Kansas State University. He has been a Fulbright Fellow in West Germany and a Visiting Associate Professor at the University of Copenhagen, and he is the founder of the Mathematical Foundations of Programming Semantics Conference/Workshop Series. His research interests include software measures, non-normal form databases, and programming semantics.

### 6.6 Robin Whitty

Dr. Robin Whitty, Ph.D., AFIMA, is currently Reader in Information Technology with the Faculty of Engineering at South Bank Polytechnic in London. He is Director of the Polytechnic's Centre for Systems and Software Engineering (CSSE). The CSSE is a research centre specializing in software measurement methods and tools, with sponsors including IBM, British Telecom, and the EEC.

Dr. Whitty was formally Lecturer in Computer Science at Goldsmiths' College University of London. During his time there he was a Vacation Fellow at British Telecom in 1987 and was Site Manager for the UK Alvey Project "Structure-based Software Metrics". He retains a visiting appointment at Goldsmiths' College as consultant to a 4-year EEC project "Cost management with Metrics of Specification (COSMOS)". He is also on the Board of Studies in Computer Science of the University of London.

Dr. Whitty's research interests include software metrics, algorithm design, and graph theory. He has 20 publications and reports in these areas, is a referee for several journals in software engineering and discrete mathematics and is a reviewer for Mathematical Reviews. He is a member of the London Mathematical Society, the European Association for Theoretical Computer Science, and the Institute of Mathematics and its Applications.