A Framework for Assessing the Adequacy and Effectiveness of Software Development Methodologies*

James D. Arthur and Richard E. Nance

Virginia Tech

Index Terms: Software Development Methodologies, Procedural Evaluation, Evaluating Methodologies, Software Engineering Objectives, Software Engineering Principles, Induced Attributes, Indicators.

1. Introduction

Over the past two decades the software development process has changed dramatically. Early software development practices were guided by "seat-of-the-pants" programming styles. Recognizing maintenance difficulties associated with such styles, the software development community began to investigate and identify software engineering principles that could significantly enhance the maintainability and quality of a resulting product. Consequently, development techniques that exploited software engineering principles like abstraction [LISB75], information hiding [PARD72] and stepwise refinement [WIRN71] were formulated and integrated into many software development processes.

The subsequent demand for increasingly complex software systems, however, mandated the *coordinated* use of *complementary* principles, guided by an encompassing software development philosophy that recognized project level goals and objectives, i.e. a *methodological* approach to software development. Today, a myriad of tools, techniques, and development methodologies address the challenging task of producing high quality software. For example, SCR [HENK78] and

* Work supported by the U.S. Navy through the Systems Research Center under Basic Ordering Agreement N60921-83-G-A165 B003-4.

DARTS [GOMH84] are development methodologies that emphasize specific software engineering goals (reducing software development costs and facilitating the design of real-time systems, respectively). SREM [ALFM85] and SADT [ROSD77] are methodology based *environments*. Both focus on particular phases of the software life cycle and are supported by unified sets of complementary tools.

The steady proliferation of design methodologies, however, has not been without a price. In particular, users find increasing difficulty in choosing an appropriate methodological approach and recognizing reasonable expectations of a design or development methodology. Addressing this concern, the research described in this paper outlines a well-defined procedure for

- evaluating the *adequacy* of a software development methodology relative to project goals, and
- assessing the *effectiveness* of a methodology relative to the quality of the product produced.

The evaluation procedure is based on a substantiated set of linkages among accepted software engineering objectives, principles, and attributes. These linkages reflect an assessment perspective structured by the needs, process, and product sequence for system development, and enable a *comparative* scale for determining the adequacy and effectiveness of the supporting development methodology. The identification of code and documentation properties and the definition of metrics for these properties enables an accumulative determination of software engineering attributes, principles and objectives.

To provide a uniform basis for discussion, Section 2 outlines the role of a methodology in the software development process. Section 3 discusses the relationship of software engineering objectives, principles and attributes to the software development effort. Section 4 identifies the commonly accepted objectives, principles and attributes, defines the relationships among them, and then discusses how one evaluates a methodology based on the those relationships. Finally, Section 5 describes an application of the assessment procedure to two Navy software development methodologies.

2. What Constitutes a Methodology

Fundamental to the research presented in this paper is a common understanding of what constitutes a "methodology". Simply stated, a methodology is a *collection* of complementary methods, and a set of rules for applying them [FREP77]. More specifically, a methodology

(1) organizes and structures the tasks comprising the effort to achieve a global objective, establishing the relationships among tasks,

(2) defines methods for accomplishing individual tasks (within the framework of the global objective), and

(3) prescribes an order in which certain classes of decisions are made, and ways of making those decisions that lead to the overall desired objective.

In general, software development methodologies should be guided by accepted software engineering principles that, when applied to the defined process, achieve a desired goal. Based on this common understanding of what constitutes a methodology, the following sections present a procedural approach to evaluating the *adequacy* and *effectiveness* of software development methodologies.

3. The Role of Objectives, Principles, and Attributes in Software Development

The development of large, complex software systems is considered a *project* activity, involving several analysts and programmers and at least one manager. What then is the role of a methodology in this setting and how does it relate to objectives, principles and attributes? Figure 1 assists in providing an answer to this question.

In general terms, an *objective* is "something aimed at or striven for." More specific to the software development context, an objective pertains to a *project* desirable – a characteristic that can be definitively judged only at the completion of the project.

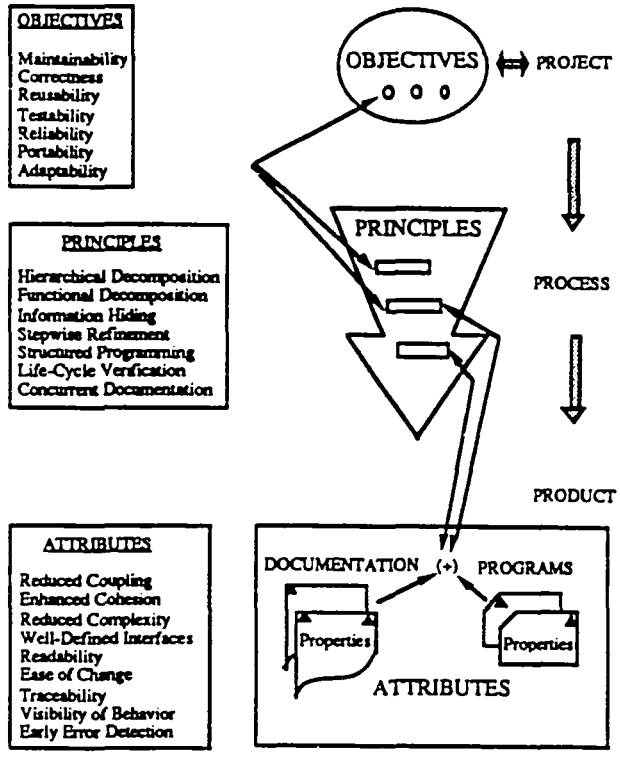


Figure 1

Illustration of the Relationships Among Objectives, Principles, Attributes in the Software Development Process

A software engineering *principle* describes an aspect of *how* the process of software development should be done. The process of software development, if it is to achieve the stipulated objectives, must be governed by these "rules of right conduct."

Attributes are the intangible characteristics of the product: the software produced by project personnel following the principles set forth by the methodology. Unlike objectives, which pertain only to the *total* project activity, attributes may be observed in one unit of the product and absent in another. The claim of presence or absence of an attribute is based on the recognition of *properties*, which contribute evidence supporting the claim. Properties are observable, and can be subjective as well as objective in nature.

Influenced by Fritz Bauer's original definition of software engineering [BAUF72] and reflecting the above description of software engineering objectives, principles and attributes, the rationale for the evaluation procedure described in this paper is founded on the philosophical argument that:

> The *raison d'etre* of any software development methodology is the achievement of one or more *objectives* through a *process* governed by defined *principles*. In turn, adherence to a process governed by those principles should result in a *product* (programs and documentation) that possesses *attributes* considered desirable and beneficial.

This philosophy, exemplified by Figure 1, is tempered by practical concerns:

(1) While a set of software engineering objectives can be identified, this set might not be complete, and additions or modification should be permitted.

(2) Objectives can be given different emphasis within a methodology or in applications of a methodology.

(3) Attributes of a large software product might be evident in one component yet missing in another.

4. A Framework for Evaluating Software Development Methodologies

A broad review of software engineering literature [BERG81, CHML90, GAFJ78, JACM75, PARD79, PARD72, SCOL78, WARJ76] leads to the identification of seven objectives commonly recognized in the numerous methodologies:

(1) Maintainability – the ease with which corrections can be made to respond to recognized inadequacies.

(2) Correctness – strict adherence to specified requirements,

(3) Reusability – the use of developed software in other applications,

(4) Testability – the ability to evaluate conformance with requirements,

(5) Reliability – the error-free performance of software over time,

(6) Portability - the ease in transferring software from one host system to another, and

(7) Adaptability – the ease with which software can accommodate to change.

The authors note that these definitions, as well as others presented in this section, are abridged: they are primarily intended to reflect a *working* understanding based on general literature usage.

Achievement of these objectives comes through the application of principles supported (encouraged, enforced) by a methodology. The principles enumerated below are extracted from the references cited above (and others) as mandatory in the creative process producing high quality programs and documentation.

(1) Abstraction - defining each program segment at a given level of refinement.

   (*a*) Hierarchical Decomposition – components defined in a top-down manner.

   (*b*) Functional Decomposition – components partitioned along functional boundaries.

(2) Information Hiding – insulating the internal details of component behavior.

(3) Stepwise Refinement – utilizing a convergent design.

(4) Structured Programming – using a restricted set of control constructs.

(5) Concurrent Documentation – management of supporting documents (system specifications, user manual, etc.) throughout the life cycle.

(6) Life Cycle Verification – verification of requirements throughout the design, development, and maintenance phases of the life cycle.

Employment of well-recognized principles should result in software products possessing attributes considered to be desirable and beneficial. A short definition of each attribute is given below.

(1) Cohesion - the binding of statements within a software component.

(2) Coupling - the interdependence among software components.

(3) Complexity - an abstract measure of work associated with a software component relative to human understanding and/or machine execution.

(4) Well-defined Interfaces - the definitional clarity and completeness of a shared boundary between a pair of components (hardware or software).

(5) Readability - the difficulty in understanding a software component (related to complexity).

(6) Ease of Change - the ease with which software accommodates enhancements or extensions.

(7) Traceability - the ease in retracing the complete history of a software component from its current status to its design inception.

(8) Visibility of Behavior - the provision of a review process for error checking.

(9) Early Error Detection - indication of faults in requirements specification and design prior to implementation.

The software development process, illustrated in Figure 1, depicts a *natural* relationship that links objectives to principles and principles to attributes. That is, one achieves the *objectives* of a software development methodology by applying fundamental *principles* which, in turn, induce particular code and documentation *attributes*. From a more detailed perspective, Figure 2 defines the *specific* set of linkages relating objectives to principles and principles to attributes. As described below, these linkages provide a framework for assessing both the *adequacy* of a methodology as well as its *effectiveness*.

4.1 Assessing the Adequacy of a Methodology

The enunciation of objectives should be the first step in the definition of a software development methodology. Closely following is the statement of principles that, employed properly, lead to the attainment of those objectives. In turn, the application of those principles within a structured software development process will yield a product that exhibits desirable attributes. The important

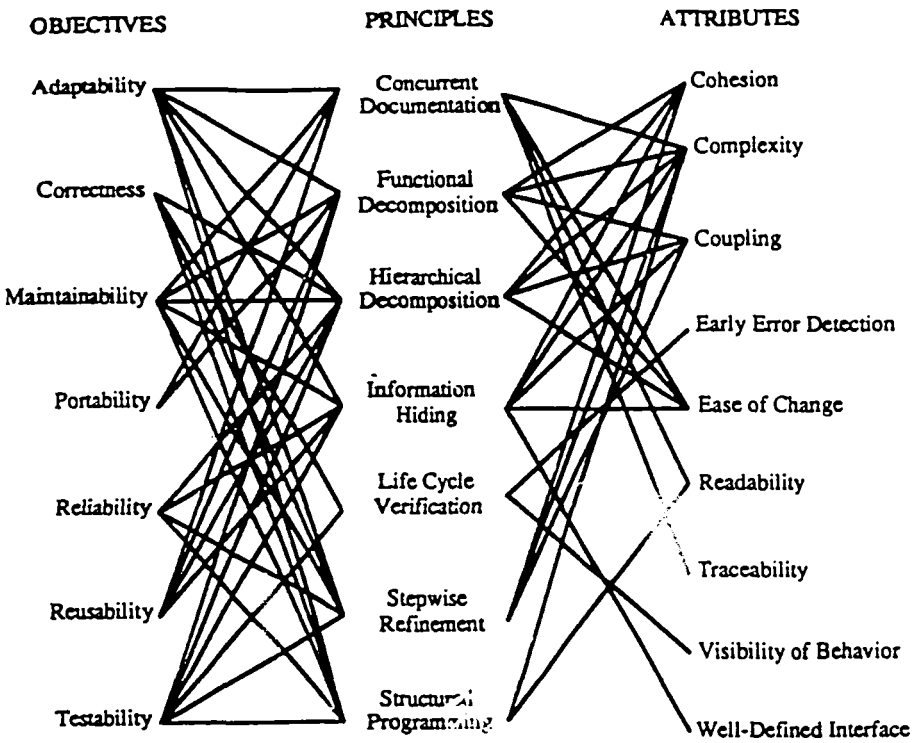


Figure 2

Linkages Among the Objectives, Principles and Attributes

correspondence between the objectives and principles and between the principles and attributes is shown in Figure 2; a literature confirmation of these relationships is discussed in [ARTJ87].

The adequacy of a software development methodology can be defined as its ability to achieve the software engineering objectives corresponding to those dictated by system needs and requirements. Intuitively, the adequacy of a methodology is assessed through a top-down evaluation scheme starting with an examination of stated methodological objectives relative to system needs and requirements. This step is then followed by a comparison of stated methodological principles and attributes with those deemed most appropriate. An examination of linkages defined by the evaluation procedure reveals the "most appropriate" set. Relative to the framework depicted by Figure 1 and the sets of linkages defined in Figure 2, an application of the evaluation procedure to the assessment of methodological *adequacy* is outlined below.

Objectives of the Methodology: The identification of *objectives* and the relationships tying objectives to needs and requirements is usually accomplished by reading the descriptions of a software development methodology. Evaluation at this level is quite subjective; however, the absence of a clear statement of objectives for a methodology should trigger an alarm: Is the "methodology" only a tool or an incomplete set of tools without coherent structure? A methodology should not be faulted, however, for emphasizing certain objectives at the expense of others; such prioritization can be highly dependent on the application domain.

Principles Defining the Process: Based on the objectives emphasized by the methodology and the predefined set of linkages among objectives and principles, the next step in assessing the adequacy of a methodology is an investigation of the software development *process*. That is, given a stated set of methodological objectives, one asks: Are the *principles* supported by the methodology consistent with those deemed necessary (through linkage examination) to achieve the stated set of objectives? The presence of principles without corresponding objective(s) or vice versa should evoke concerns. Although this level of evaluation is inherently subjective, some analytical quality is introduced through the established objective/principle correspondence.

Attributes of the Product: The third step in the assessment process, formulating the set of *expected* product attributes, is based on the fact that principles govern the process by which a software product is produced. That is, a given set of principles should induce a consequent set of product attributes. Obviously, the expected set of product attributes should correspond to those desired by the software engineer, and to some extent, be implied or stated in the description of the software development methodology. More objectivity is introduced at this level because, although intangible, evidence of the attributes should be discernible in the product.

4.2 Assessing the Effectiveness of a Methodology

While a top-down evaluation process reveals deficiencies of a software development methodology, the *effectiveness* of a methodology is assessed through a bottom-up evaluation process. As

the term implies, the effectiveness of a methodology is defined as the degree to which a methodology produces a desired result. In particular, the effectiveness of a methodology is reflected by a product's conformance to the software development process defined by that methodology. We note, however, that elements *independent* of the methodology can influence its effectiveness, e.g. an inadequate understanding and/or use of the methodology.

The Existence of Product Attributes: Assessing methodological effectiveness starts with an examination of the software product (code and documentation) for the presence or absence of attributes. Because attributes are intangible, subjective qualities, the current evaluation is based on defined properties that provide evidence as to the presence or absence of attributes. More specifically, the computation of metric values reflect the extent to which particular properties are observed. In turn, this information is used to synthesize the extant set of product attributes. Clearly, the set of attributes determined from a product evaluation should agree with those induced by the corresponding development methodology. Set mismatch can imply an inappropriate software development methodology, an inadequate understanding of the methodology, or perhaps, the failure of users to adhere to the principles advocated by the methodology.

Implied Principles and Objectives: Knowing which attributes are present in the product, and the extent to which they are assessed present, provides a basis for implying the use of software engineering principles in the software development process. The rationale for such a statement is based on the observation that a principle-to-attribute linkage conversely indicates an attribute-to-principle relationship. Implying principle usage must be tempered, however, because of the many-to-one relationships existing between attributes and principles. Similarly, using the established linkages among objectives and principles, one can speculate on the achievement of stated software engineering objectives.

In summary, the three levels of examination defined by top-down evaluation process establishes a procedure for determining how well a methodology can support perceived needs, requirements, and the software development process. Conversely, the bottom-up evaluation process reveals how

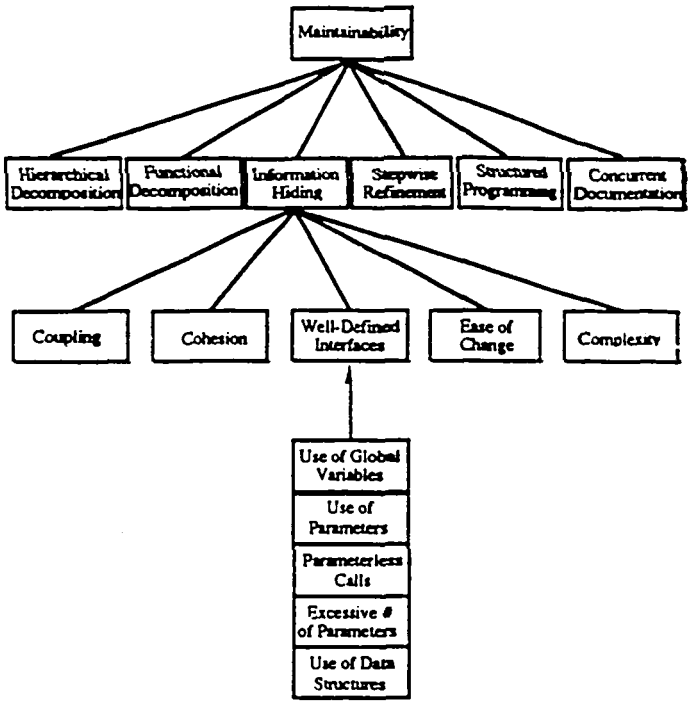


Figure 3

Illustration of the Evaluation Process

well the methodology is applied in the software development process through the use of *quantitative* measures to support an objective, *qualitative* assessment.

4.3. An Illustration of the Evaluation Process

To illustrate how the evaluation scheme can be applied, we refer the reader to Figure 3 while considering the single objective of *maintainability*. Formally, maintainability can be defined as the ease with which maintenance can be performed to a functional unit in accordance with prescribed requirements. Accepting maintainability as an objective mandates the inclusion of six principles (hierarchical decomposition, functional decomposition, information hiding, stepwise refinement, structured programming and concurrent documentation) contributing to the realization of that

objective. That is, if a methodology emphasizes maintainability as an objective, then it should also stress the use of the six principles that are related to maintainability.

Expanding on one of those principles, information hiding, we note the five attributes (reduced coupling, enhanced cohesion, well-defined interfaces, ease of change, and low complexity) that should be evident in software developed using a process governed by the principle of information hiding. This set of expected attributes is then compared to the desired set for correspondence.

In general, a methodology should emphasize the same set of software engineering objectives derived from project level requirements. The methodology should correspondingly stress the set of principles linked to those objectives. Additionally, the expected set of product attributes (defined by the linkages among principles and attributes) should agree with the set deemed most desirable by the project manager. If the above conditions are met, then the candidate methodology is assumed to be adequate relative to project level, software engineering objectives.

On assessing the effectiveness of an methodology, let us first observe the relationship between a particular attribute and specific product characteristics. Referring again to Figure 3, and narrowing our attention to one of the attributes, well-defined interfaces, we identify one set of characteristics related to the well-defined interfaces attribute. These characteristics form the set of *observable* properties which contribute to the claim that a piece of software exhibits a well-defined interface. Although the properties shown are only a subset of those previously identified [ARTJ86], they represent *both* confirming and contrasting elements. For example, the use of global variables has a negative impact on well-defined interfaces. The use of structured data in parameter calls, however, has a positive impact.

Hence, to determine the effectiveness of a methodology one assesses the extent to which product attributes are present (or absent), and then propagate the results of that assessment through the sets of linkages defined by the evaluation procedure. As discussed by Kearney [KEAJ86], however, that assessment process must be predicated on validated metrics.

5. Application of the Evaluation Procedure

Based on the defined set of linkages among objectives, principles, and attributes, the operational specification of the evaluation procedure is guided by two fundamental axioms:

(1) the methodology description and project requirements provide standards, conventions, and guidelines that *describe how to produce* a product, and

(2) the project documentation, code, and code documentation *reflect how well* the development process prescribed by the methodology is followed.

As described below, an application of the evaluation procedure, guided by the above two axioms, illustrates the utility and intrinsic prower of the evaluation procedure in assessing the adequacy and effectiveness of a methodology. Provided in this illustration is a characterization of the components used in the evaluation process, an individual assessment of two methodology descriptions, an analysis of associated products, and a summary of the results. The authors note that a substantial part of the characterization and assessment process is embodied in the operational aspects of applying the evaluation procedure. Length restrictions, however, prevent their discussion. For information on the operational aspects the authors refer the interested reader to [ARTJ86, ARTJ87].

5.1 Data Sources

A joint investigation of two comparable Navy software development methodologies and respective products is detailed in [NANR85]. The investigation effort utilizes:

- Four software development methodology documents for

  (1) identifying the pronounced software engineering objectives, principles, and attributes, and

(2) assessing the adequacy of each methodology through the objective/principle/attribute linkages defined by the evaluation procedure, and

- Eight software system documents and 118 routines, comprising 8300 source lines of code, for

  (1) determining the evident set of product attributes, and

  (2) via the attribute/principle/objective linkages, empirically assessing the principles and objectives emphasized during product development.

The following section provides a summary of the results and illustrates the utility and versatility of the procedural approach to evaluating software development methodologies. For simplicity, we refer to the software systems as system A and system B (and methodology A, methodology B, respectively).

5.2 Analyzing the Methodological Description and Associated Product

The initial step in the evaluation process is to perform a "top-down" analysis of methodologies A and B, to reveal the set of *claimed* software engineering objectives, principles, and attributes. Because both methodologies have experienced evolutionary development, a *clear* statement of their respective methodological objectives is lacking. Nonetheless, as detailed in Figure 4, the documentation for methodology A does appear to stress the objectives of *reliability* and *correctness* supported by the principles of *structured programming, hierarchical decomposition,* and *functional decomposition.* Following the objective/principle relationships defined by the evaluation procedure, for each objective stressed in methodology A only three of the necessary six principles are emphasized. The implication is, that unless the principles of life-cycle verification, information hiding and stepwise refinement are implicitly assumed *and* utilized, correctness and reliability are compromised.

	Methodology A	Methodology B
Objectives		
Maintainability		Yes
Correctness	Yes	
Reusability		
Testability		
Reliability	Yes	Yes
Portability		
Adaptability		Yes
Principles		
Hierarchical Decomposition	Yes	
Functional Decomposition	Yes	
Information Hiding		
Stepwise Refinement		
Structured Programming	Yes	Yes
Concurrent Documentation		Yes
Life Cycle Verification		
Attributes	None	None

Figure 4

Pronounced Objectives, Principles, and Attributes

Using metric values and properties, a corresponding "bottom-up" examination of product A provides some interesting results. The Kiviat graph displayed in Figure 5a illustrates the extent to which each attribute is *assessed* as present in the product. (Attribute ratings are restricted to an arbitrarily chosen 1-10 scale.) Note that (reduced) complexity attains the highest rating – 8.0 out of 10.0, closely followed by readability (7.4) and cohesion (6.8). Based on the three principles stressed in methodology A, the evaluation procedure predicts that (reduced) complexity, readability, and cohesion *should*, in fact, be among the product attributes.

In concert with the stated objectives and principles for methodology A, Figure 5b reveals that structured programming (7.7) is the prominent principle used in developing system A, followed by stepwise refinement (6.7), hierarchical decomposition (6.4), and functional decomposition (6.4). Figure 5c depicts the results of emphasizing these principles in terms of methodology objectives. In

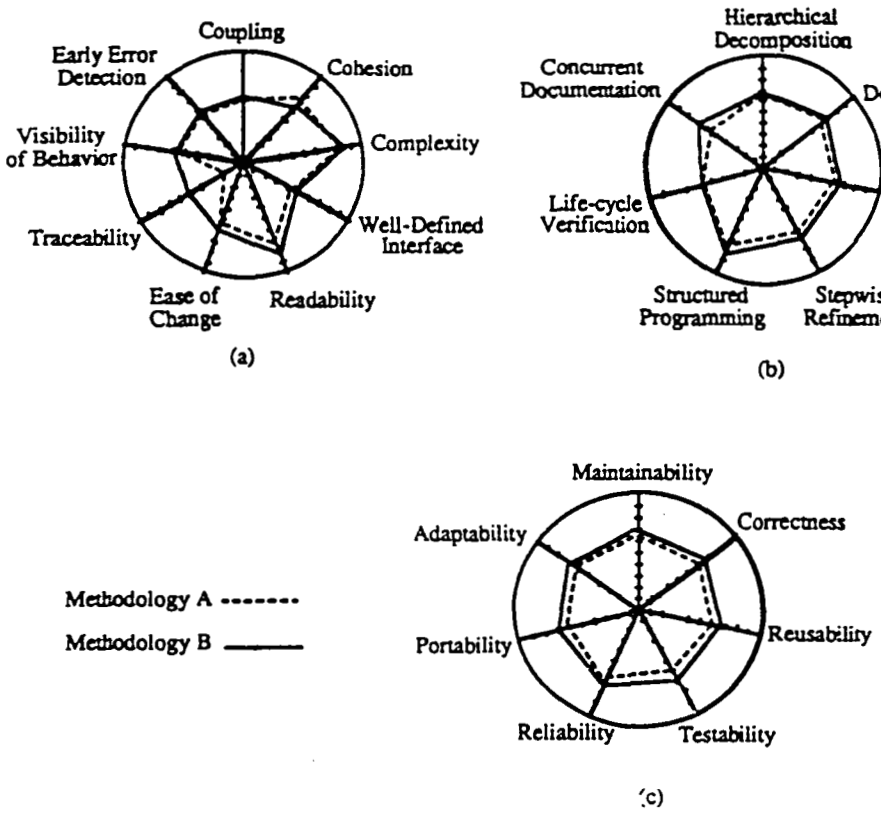


Figure 5

Detected Presence of Objectives, Principles, and Attributes

particular, reliability is rated as the major software development objective (6.7). Although correctness is also stressed by methodology A, ascertaining correctness necessitates life-cycle verification. This principle is neither emphasized by methodology A, nor evident in the software product. As illustrated by Figures 5a, 5b and 5c, other objectives and principles are given some emphasis during the software development process for system A. It is the authors' opinions, however, that because they are not explicitly stressed in methodology A, the associated product suffers.

For methodology B, the objectives enunciated in the documentation are *maintainability*, *adaptability*, and *reliability*. *Structured programming* and *concurrent documentation* are the emphasized principles. Like methodology A, however, a complete set of supporting principles are not stated.

Hierarchical decomposition, functional decomposition, and to some extent *information hiding* are implicitly assumed as underlying principles of methodology B. According to the linkages among objectives and principles, all of the above principles (both stated and assumed) are required to achieve the objectives explicitly stated in methodology B.

Subsequent analysis of product B and a "bottom-up" propagation of the results through the linkages defined by the evaluation procedure reveals structured programming as the most prominent principle (8.3), closely followed by concurrent documentation (7.0). Moreover, the evaluation also indicates that the implicitly assumed principles of methodology B are utilized – stepwise refinement, hierarchical decomposition, functional decomposition, and information hiding rate 6.9, 6.7, 6.7, and 6.3, respectively. Finally, the results imply that during the development of product B the objectives of maintainability, adaptability, and reliability are most emphasized. The above assessments are illustrated in Figures 5a, 5b, and 5c.

To summarize, the evaluation procedure reveals that both methodologies lack a clear statement of goals and objectives, as well as sufficient principles for achieving the objectives that are emphasized. Moreover, glaring deficiencies are apparent in both software development methodologies. That is, both fail to actively support the principle of information hiding and also have difficulties in incorporating the desirable attributes of traceability and well-defined interfaces in respective system products. In general, the evaluation procedure does accurately assesses the software engineering objectives, principles, and attributes *espoused* by methodologies A and B. Of particular significance, however, is that the objectives and principles *determined* to be "emphasized" during the product development process, yet not stated in the methodology documentation, are precisely those that are *implicitly assumed* important by the software engineers developing products A and B. A more detailed account of the evaluation can be found in [NANR85].

6. Conclusion

Tools, techniques, environments, and methodologies dominate the software engineering literature, but relatively little research in the evaluation of methodologies is evident. This work

reports an initial attempt to develop a procedural approach to evaluating software development methodologies. Prominent in this approach are:

(1) an explication of role of a methodology in the software development process,

(2) the development of a procedure based on linkages among objectives, principles, and attributes, and

(3) the establishment of a basis for reduction of the subjective nature of the evaluation through the introduction of properties.

An application of the evaluation procedure to two Navy methodologies has provided consistent results that demonstrate the utility and versatility of the evaluation procedure [NANR85]. Current research efforts focus on the continued refinement of the evaluation procedure through

(a) the the identification and integration of product quality *indicators* reflective of attribute presence, and

(b) the *validation* of metrics supporting the measure of those indicators.

The consequent refinement of the evaluation procedure offers promise of a flexible approach that admits to change as the field of knowledge matures. In conclusion, the procedural approach presented in this paper represents a promising path toward the end goal of objectively evaluating software engineering methodologies.

References

[ALFM85] Alford, M., "SREM at the age of Eight; The Distributed Computing Design System," *IEEE Computer*, Vol. 18, No. 4, April 1985, pp. 36-54.

[ARTJ86] Arthur, J.D., Nance, R.E. and Henry, S.M., "A Procedural Approach to Evaluating Software Development Methodologies: The Foundation," Technical Report TR-86-24, Virginia Tech, 1986.

[ARTJ87] Arthur, J.D. and Nance, R.E., "Developing an Automated Procedure for Evaluating Software Development Methodologies and Associated Products," Technical Report SRC-87-007, Systems Research Center, Virginia Tech, 1987.

[BAUF72] Bauer, F.L. "Software Engineering," *Information Processing 71*, North Holland Publishing Company, 1972.

[BERG81] Bergland, G.D. "A Guided Tour of Program Design Methodologies," *Computer*, Vol. 14, No. 10, October 1981, pp. 13-36.

[CHML90] Chmura, L.J., Norcio, A.f., and Wicinski, T.J., "Evaluating Software Design Processes by Analyzing Change Data Over Time," *IEEE Transactions on Software Engineering*, Vol. 16, No. 7, July 1990, pp. 729-740.

[FREP77] Freeman, P., "The Nature of Design," *A Tutorial on Software Design Techniques*. Second edition, IEEE Computer Society Press, 1977, pp. 29-36.

[GAFJ81] Gaffeney, J. E., "Metrics in Software Quality Assurance," *Proceedings of the National ACM Conference*, November 1981, pp. 126-130.

[GOMH84] Gomaa, H. "A Software Design Method for Real-Time Systems," *Communications of the ACM*, Vol. 27, No. 9, September 1984, pp. 938-949.

[HENK78] Heninger, K. L., J. W. Kallander, J. E. Shore, and D. L. Parnas, "Software Requirements for the A-7E Aircraft," NRL Memorandum Report 3876, Naval Research Laboratory, Washington, D. C., November, 1978.

[JACM75] Jackson, M., *Principles of Program Design*, London: Academic Press. 1975.

[KEAJ86] Kearney,J., *et. al.*, "Software Complexity Measurement," *Communications of the A.C.M.*, Vol.29, No. 11, November 1986, pp. 1044-1050. Monterey, CA, March 1987, pp. 238-252.

[LISB75] Liskov, B., Zilles, S., "Specification Techniques For Data Abstraction," *IEEE Transactions on Software Engineering* , Vol. SE-1, No. 1, March 1975, pp. 7-19.

[NANR85] Nance, R.E., Arthur, J.D. and Dandekar, A.V. "Evaluation of Software Development Methodologies," A Final Report of the Immediate Software Development Project, The Department of Computer Sciences, Virginia Tech, December 1985.

[PARD76] Parnas, D., "On the Design and Development of Program Families," *IEEE Transactions on Software Engineering*, Vol. SE-2, No. 1, March 1976, pp. 1-9.

[PARD72] Parnas, D., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM* , Vol. 15, No. 5, May 1972, pp. 330-336.

[ROSD77] Ross, D., "Structured Analysis: A Language for Communicating Ideas," *IEEE Transactions on Software Engineering*, Vol. SE-3, No. 1, January, 1977, pp. 16-34.

[SCOL78] Scott, L., "An Engineering Methodology for Presenting Software Functional Architecture," *Proceedings of the Third International Conference on Software Engineering* , NY, 1978, pp.222-229.

[WARJ76] Warnier, J. *Logical Construction of Programs*, 3rd edition, trans. B. Flanagan, NY: Van Nostrand Reinhold, 1976.

[WIRN71] Wirth, N., "Program Development by Stepwise Refinement," *Communications of the ACM*, Vol. 14, No.4, April, 1971, pp. 221-227.

VIEWGRAPH MATERIALS

FOR THE

R. NANCE PRESENTATION

A FRAMEWORK FOR ASSESSING THE ADEQUACY AND EFFECTIVENESS OF SOFTWARE DEVELOPMENT METHODOLOGIES

A Presentation to the
Fifteenth Annual Software Engineering Workshop

Richard E. Nance
James D. Arthur

Systems Research Center
and
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia

28 November 1990

THE ORIGIN

Immediate Software Development Issues for Embedded Systems Applications in Surface Combatants (25 March - 15 September 1985)

Issue: Multiple Software Development Methodologies

(a) *Review* two software development "methodologies" (A and B)

(b) *Compare* and *evaluate* A and B

(c) *Assess* the costs and benefits of:

- Continuing with multiple software development methodologies
- Using only one software development methodology
- Transitioning to an alternate software development methodology

A	==>	B
B	==>	A
A/B	==>	???

OUTLINE

Evaluation Approach

- Objectives, Principles, Attributes Framework

Development of an Evaluation Procedure

- Software Engineering and Software Development
- A Structured Evaluation Procedure
- Data Sources

Application of the Evaluation Procedure

- Summary of Sample Data
- Illustration of Procedure Application

Summary of Results

Future Work

EVALUATION APPROACH

1. Develop an Evaluation Procedure
   - What is a methodology?
   - How can they be compared?

2. Apply the Evaluation Procedure
   - In consonance with our Navy sponsor, and with
   - Contributions from software development sites and oversight agencies.

ON METHODOLOGIES

What is a methodology?

A methodology is a collection of complementary methods, and a set of rules for applying them. More specifically, a methodology

(1) organizes and structures tasks comprising the effort to achieve a global objective, establishing the relationships among tasks,

(2) defines methods for accomplishing individual tasks (within the context of the global objective), and

(3) prescribes an order in which certain classes of decisions are made, and ways of making those decisions that lead to the desired objective.

RATIONALE FOR THE EVALUATION PROCEDURE

A set of objectives can be identified that include those postulated by any software engineering methodology. A methodology defines those principles that characterize a proper and appropriate development process. Adherence to a process governed by these principles should result in a product (programs and documentation) that possesses attributes considered desirable and beneficial.

Philosophy tempered by practical concerns:

(1) Sets of objectives, principles, attributes are open.

(2) Prioritization of objectives recognized.

(3) Components of large software system vary – attribute sampling.

(4) Flexible application of evaluation procedure – consonant with project objectives.

FRAMEWORK FOR SOFTWARE DEVELOPMENT

OBJECTIVES

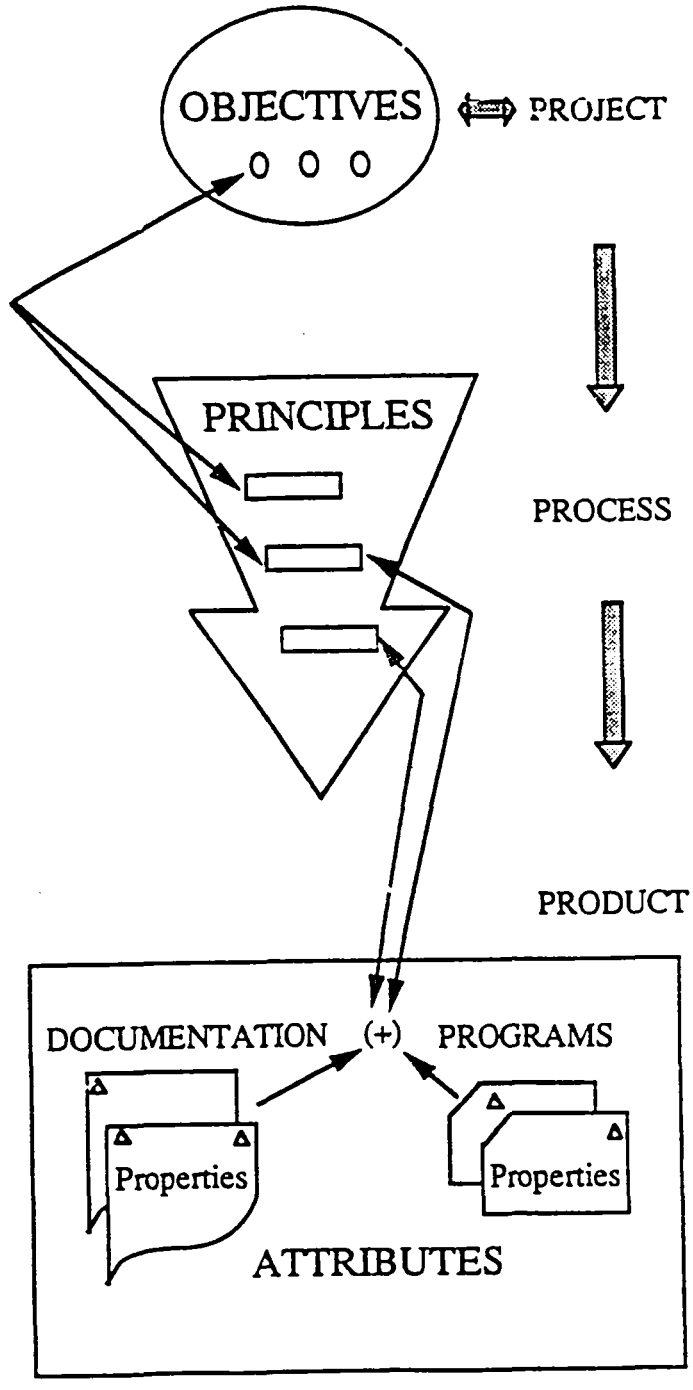
Maintainability
Correctness
Reusability
Testability
Reliability
Portability
Adaptability

PRINCIPLES

Hierarchical Decomposition
Functional Decomposition
Information Hiding
Stepwise Refinement
Structured Programming
Life-Cycle Verification
Concurrent Documentation

ATTRIBUTES

Reduced Coupling
Enhanced Cohesion
Reduced Complexity
Well-Defined Interfaces
Readability
Ease of Change
Traceability
Visibility of Behavior
Early Error Detection



PROCEDURE DEVELOPMENT

1. Identify Objectives

   - What qualities are desirable?

2. Define Principles

   - How are desirable qualities obtained?

3. Link Principles to Objectives

   - Which principles contribute to each objective?

4. Define Resulting Attributes

   - Use of a principle induces what desirable attributes?

5. Define Properties Associated with Attributes

   - What properties give evidence of attribute presence or absence?

   - How to measure properties?

PRIMARY SOFTWARE ENGINEERING OBJECTIVES

(1) **Adaptability** - the ease with which software can accommodate to changing requirements

(2) **Correctness** - strict adherence to specifications

(3) **Maintainability** - the ease with which corrections can be made to respond to recognized inadequacies

(4) **Portability** - the ease in transferring software to another host environment

(5) **Reliability** - the error-free behavior of software over time

(6) **Reusability** - the use of developed software in other applications

(7) **Testability** - the ability to evaluate conformance with specifications

PRIMARY SOFTWARE ENGINEERING PRINCIPLES

(1) **Abstraction** - defining each program segment at a given level of refinement

(a) *Hierarchical Decomposition* - components defined in a top-down manner

(b) *Functional Decomposition* - components partitioned along functional boundaries

(2) **Concurrent Documentation** - management of supporting documents (system specifications, user manuals, etc) throughout the life cycle

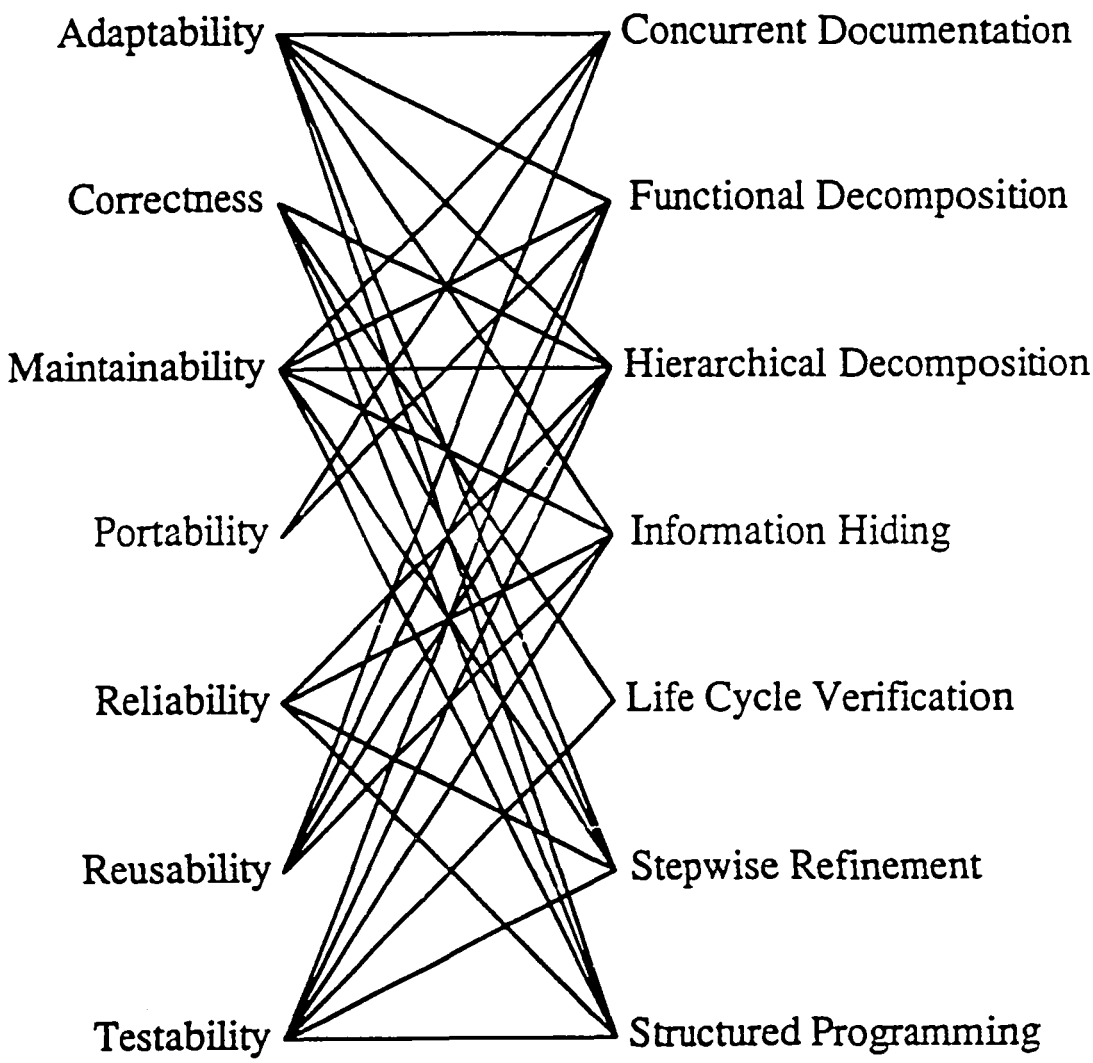
(3) **Information Hiding** - insulating the internal details of component behavior

(4) **Life Cycle Verification** - verification of requirements throughout the design, development, and maintenance phases of the life cycle

(5) **Stepwise Refinement** - utilizing convergent design

(6) **Structured Programming** - using a restricted set of program control constructs

OBJECTIVES / PRINCIPLES LINKAGES



PRIMARY SOFTWARE ENGINEERING ATTRIBUTES

(1) **Cohesion** - The binding of statements within a software component

(2) **Complexity** - an abstract measure of work associated with a software component

(3) **Coupling** - the interdependence among software components

(4) **Early Error Detection** - indication of faults in requirements, specification and design prior to implementation

(5) **Ease of Change** - software that accommodates enhancements or extensions

(6) **Readability** - the difficulty in understanding a software component

(7) **Traceability** - the ease in retracing the complete history of a software component from its current status to its design

(8) **Visibility of Behavior** - the provision of a review process for error checking

(9) **Well-Defined Interfaces** - the definitional clarity and completeness of a shared boundary between software and/or hardware (software/software, software/hardware)

PRINCIPLES / ATTRIBUTES LINKAGES

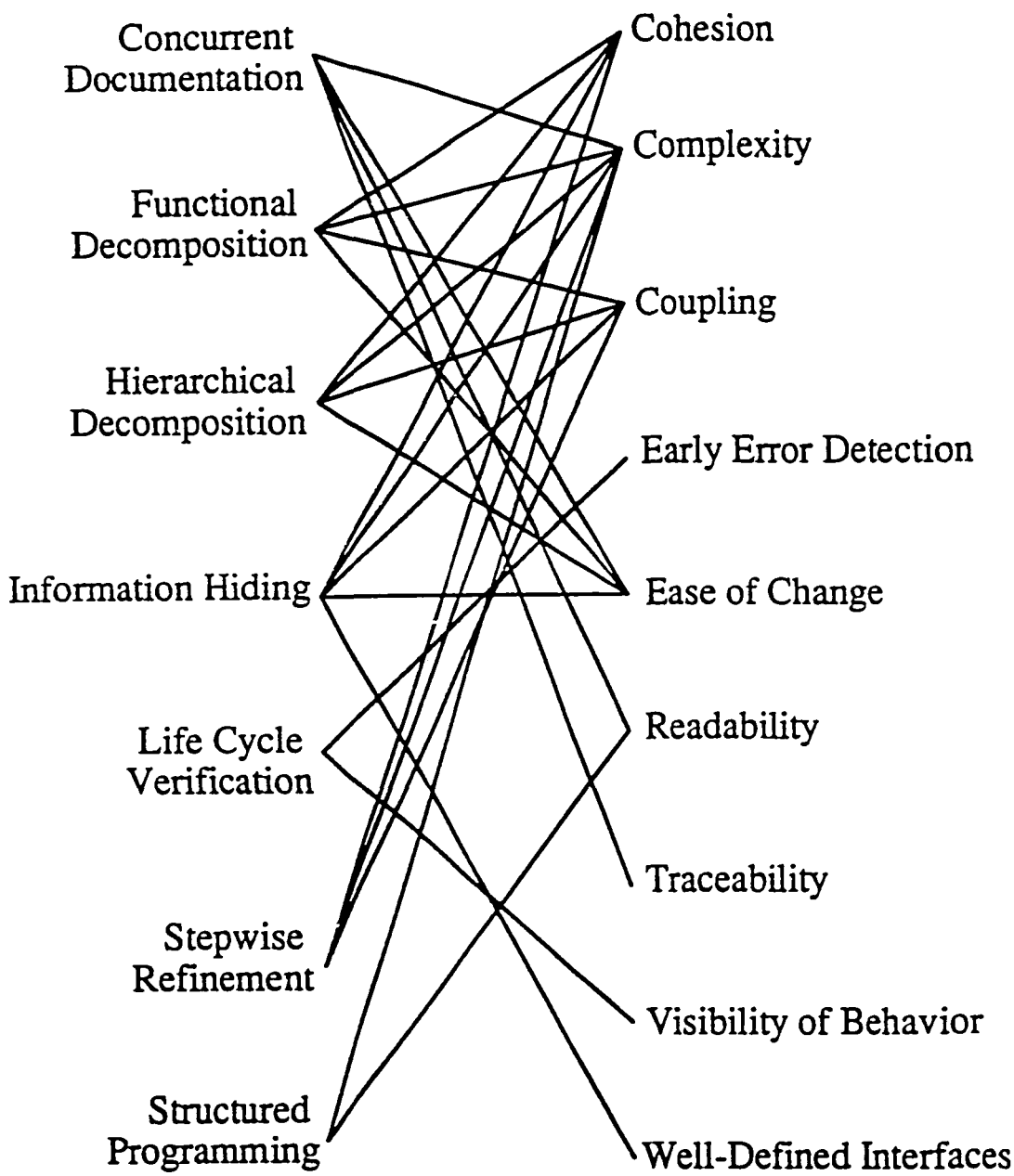
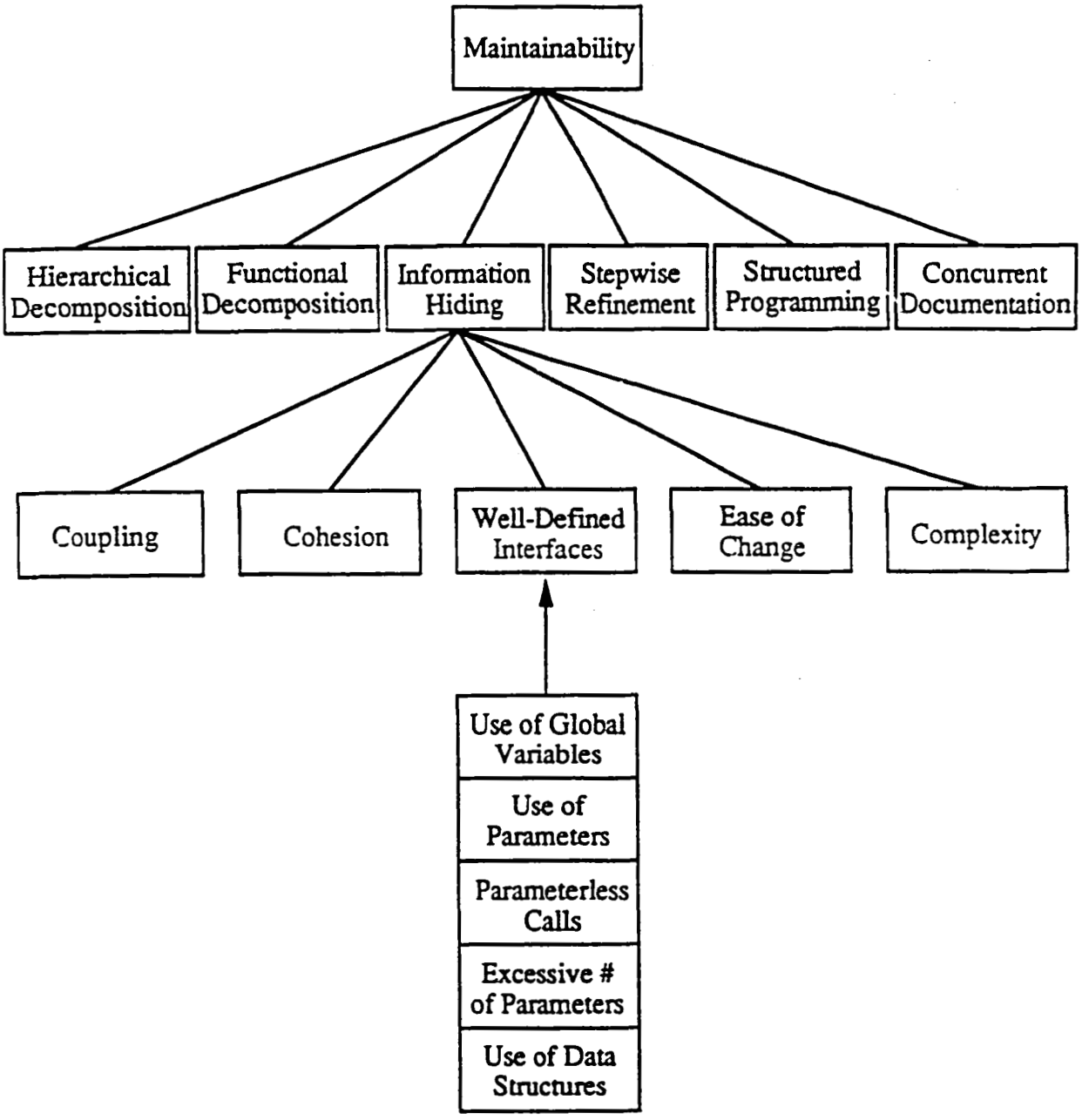
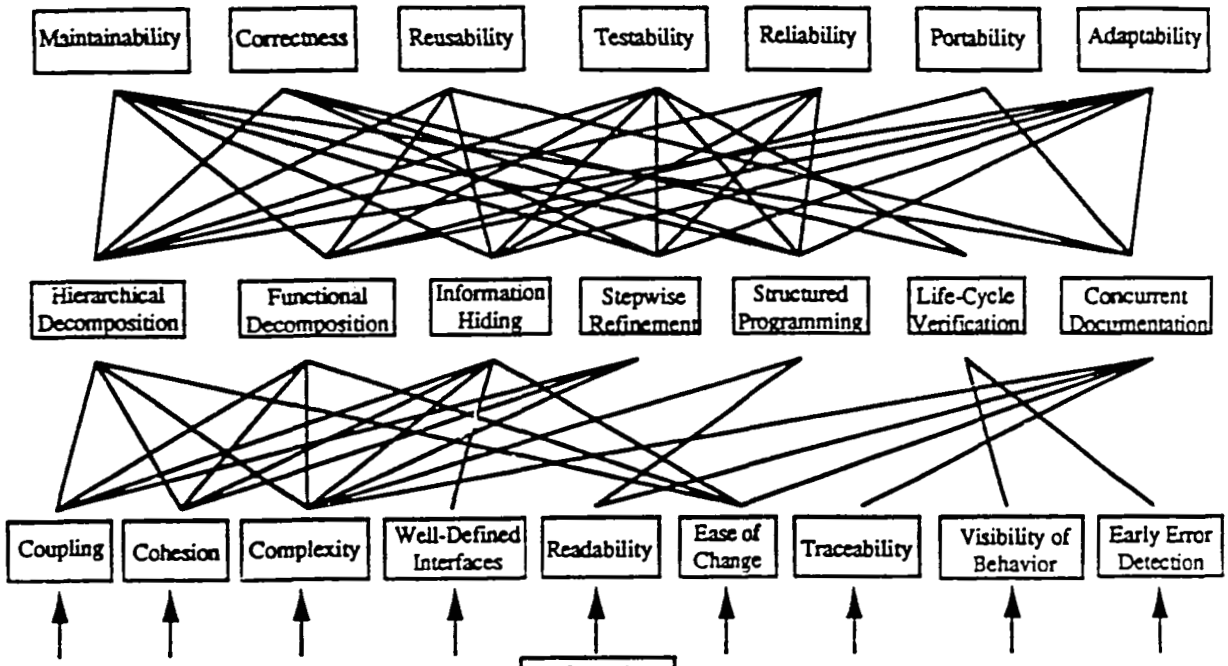


ILLUSTRATION OF THE EVALUATION PROCEDURE



SETS OF DEFINED LINKAGES



		Control Structures	(+)
		GOTO's	(-)
(+)	Special Characters	Code Indentation	(+)
(+)	Symbolic Constants	# Block Comments	(+)
(+)	Documentation Readability	Meaningful Names	(+)
(+)	Completion & Accuracy of Documentation	# Single Line Comments 'In Line'	(-)
(-)	Embedded Alternate Language	Correct Grammar & Spelling	(+)
(+)	Short Modules	Consistent Comments	(+)

THE OPA FRAMEWORK FOR EVALUATION: SUMMARY

Fundamental to the evaluation procedure are several sets of linkages:

Linkages	Defined	Substantiated
• Objectives / Principles	(33)	(33)
• Principles / Attributes	(24)	(24)
• Attributes / Properties	(125)	(114)
66 Automatable		

Assessing the *adequacy* of a methodology is achieved through a "top-down" evaluation process.

Assessing the *effectiveness* of a methodology is achieved through a "bottom-up" evaluation process.

APPLICATION OF THE PROCEDURE: SUMMARY OF SAMPLE DATA

Documents (Primary)

A: The Combat System Development Plan

The Computer Programming Manual

The Program Development Manual

Six Numbered Documents (PDS, IDS)

B: Functional Description Document

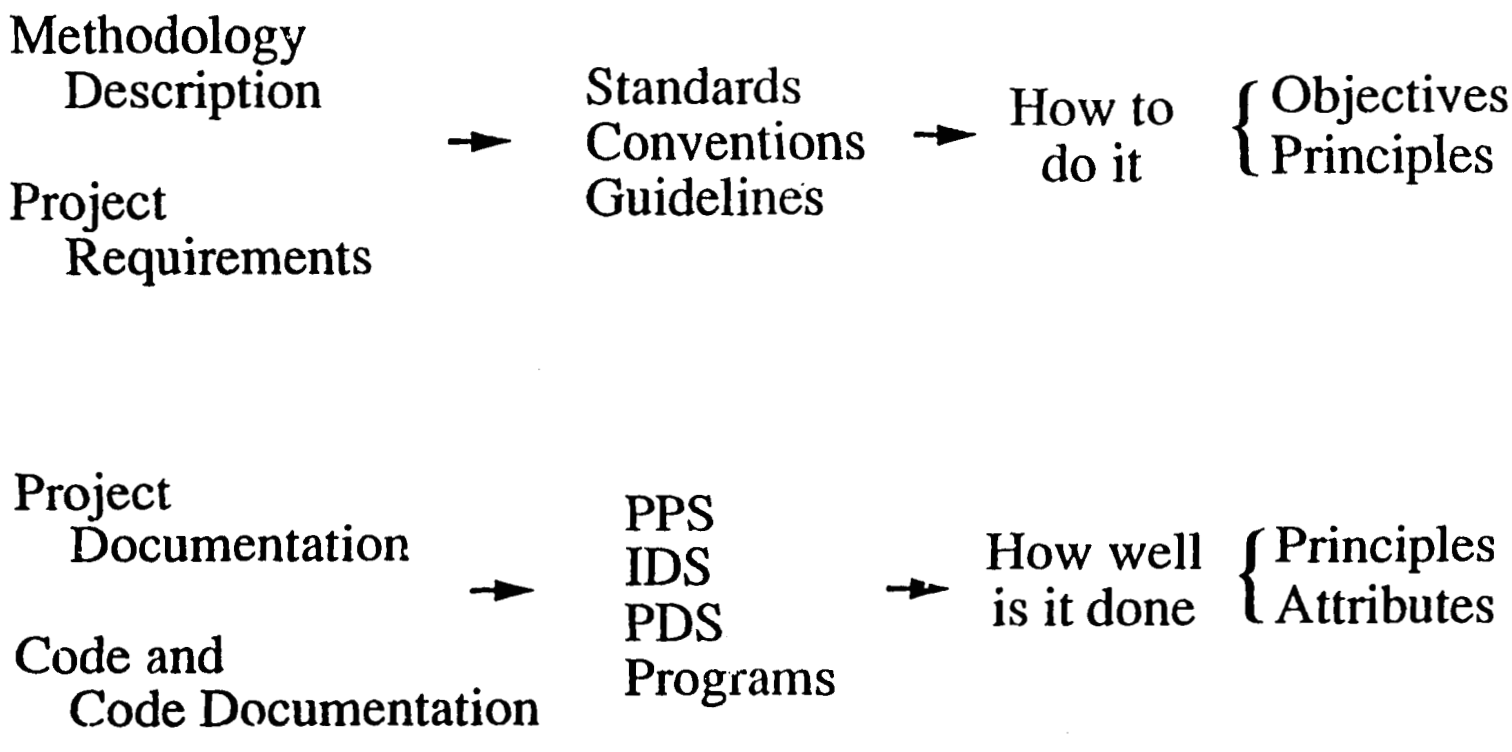
Two Numbered Documents (PDS, IDS)

Source Code:

A:	Routines = 17	SLOCS = 1170
	SysProcs = 2	SLOCS = 1370
B:	Routines = 99	SLOCS = 5729

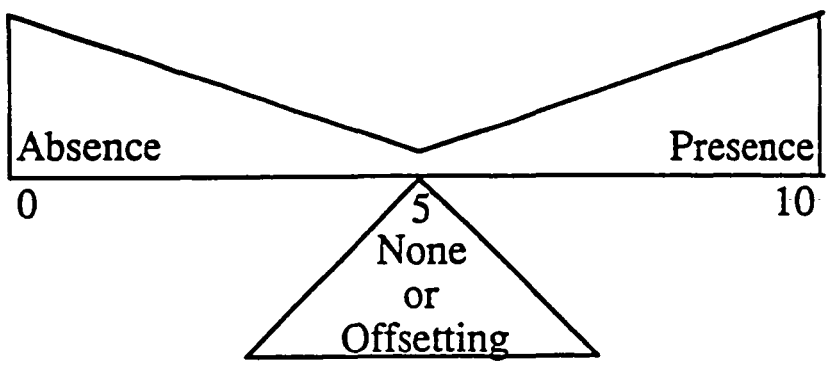
R. Nance
VPI

DATA SOURCES AND IMPLICATIONS



AN ACCUMULATION OF EVIDENCE

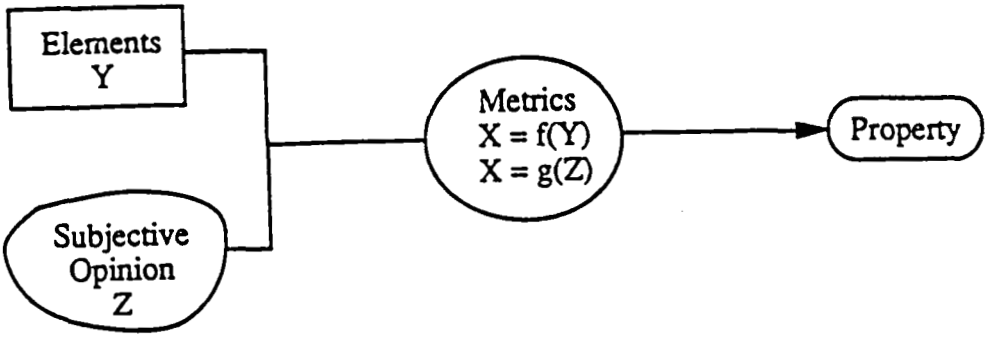
"Demonstrating that software possesses a desired attribute (or does not) is not a proof exercise; rather, it resembles an exercise in civil litigation in that evidence is gathered to support both contentions (the presence or absence) and weighed on the scales of comparative judgement."



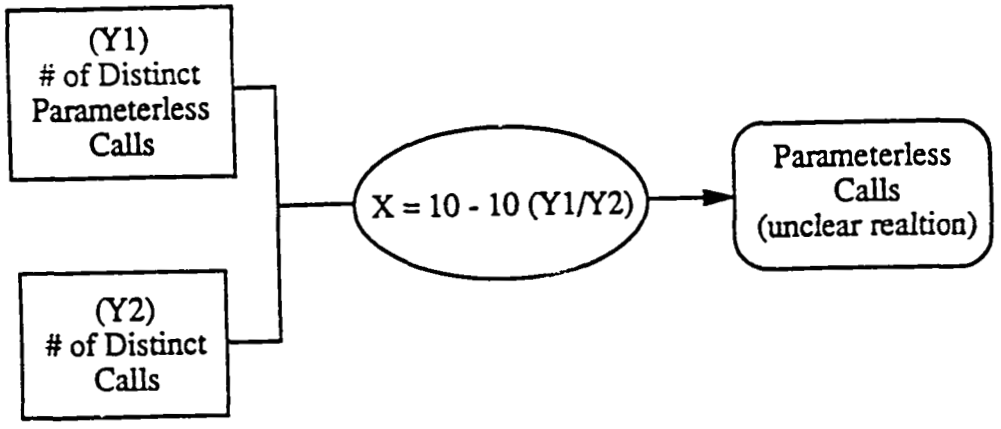
Measurement Scale

ELEMENTS, METRICS AND PROPERTIES

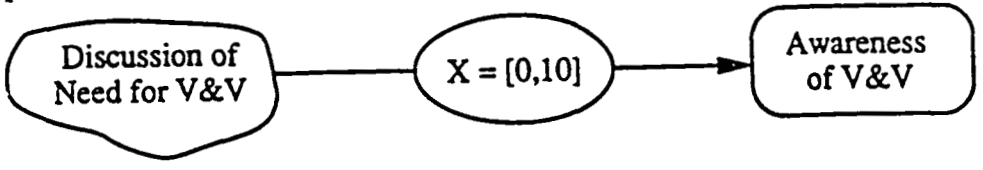
- Relationship



- Code Example



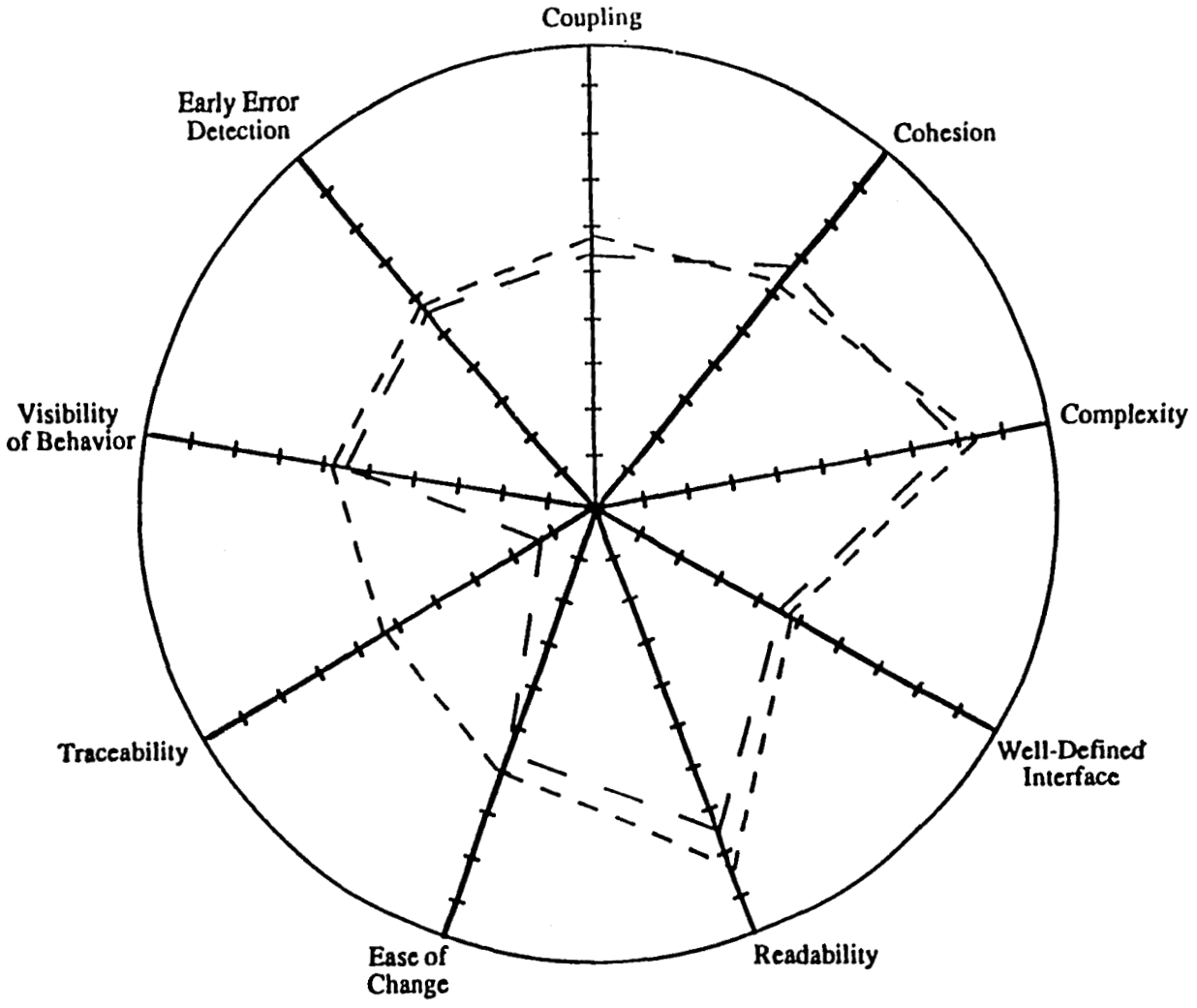
- Documentation Example



ASSESSING "METHODOLOGICAL" EFFECTIVENESS (ATTRIBUTES)

	A	B
Coupling	5.4	5.9
Cohesion	6.8	6.4
Complexity	8.0	8.4
Well-Defined Interfaces	4.7	4.8
Readability	7.4	8.2
Ease of Change	5.6	6.0
Visibility of Behavior	5.6	5.8
Early Error Detection	5.6	5.8
Traceability	1.2	5.3

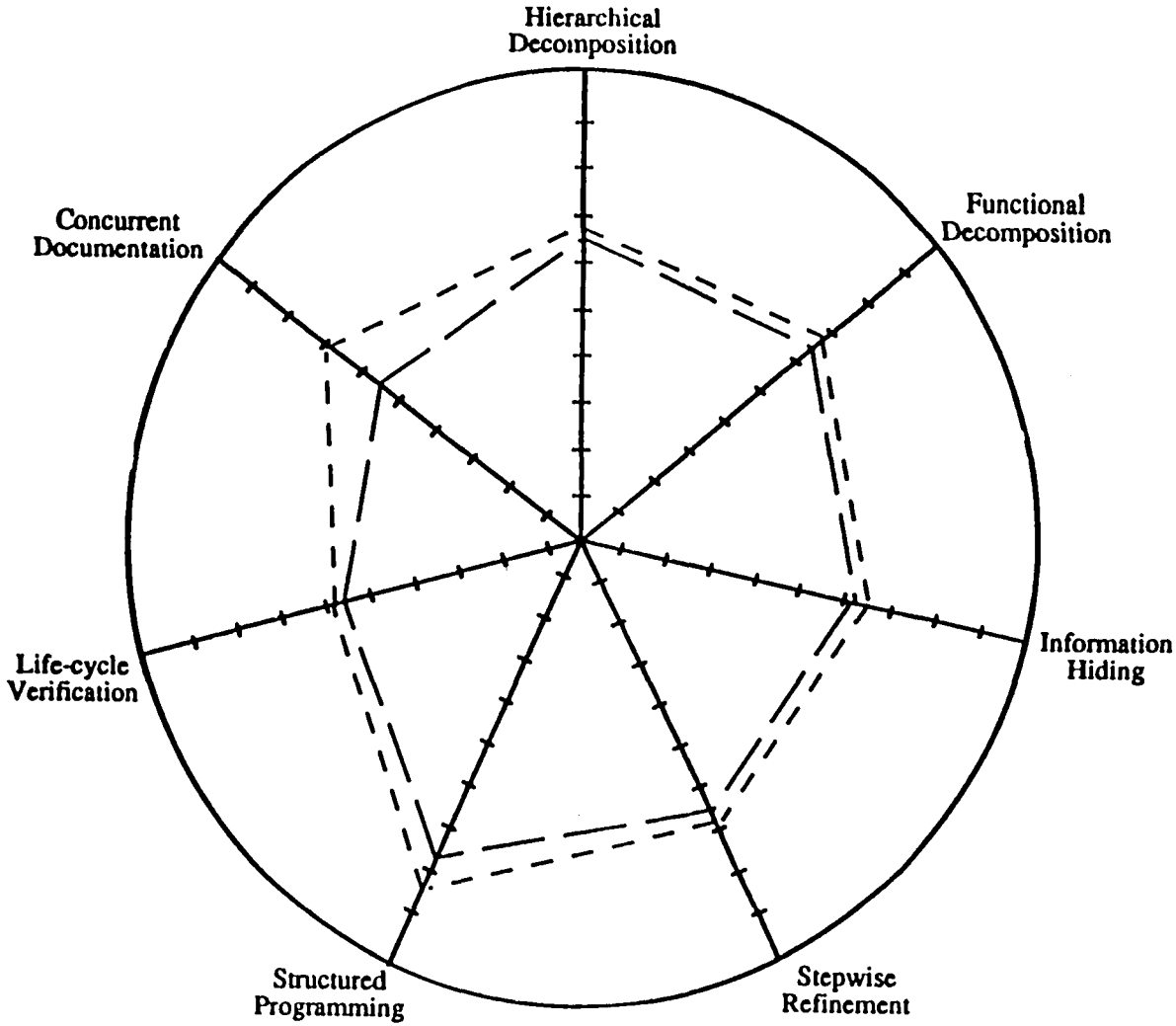
Both have difficulty with Traceability and Well-Defined Interfaces



KIVIAT GRAPH FOR ATTRIBUTES

Methodology A — — — — —

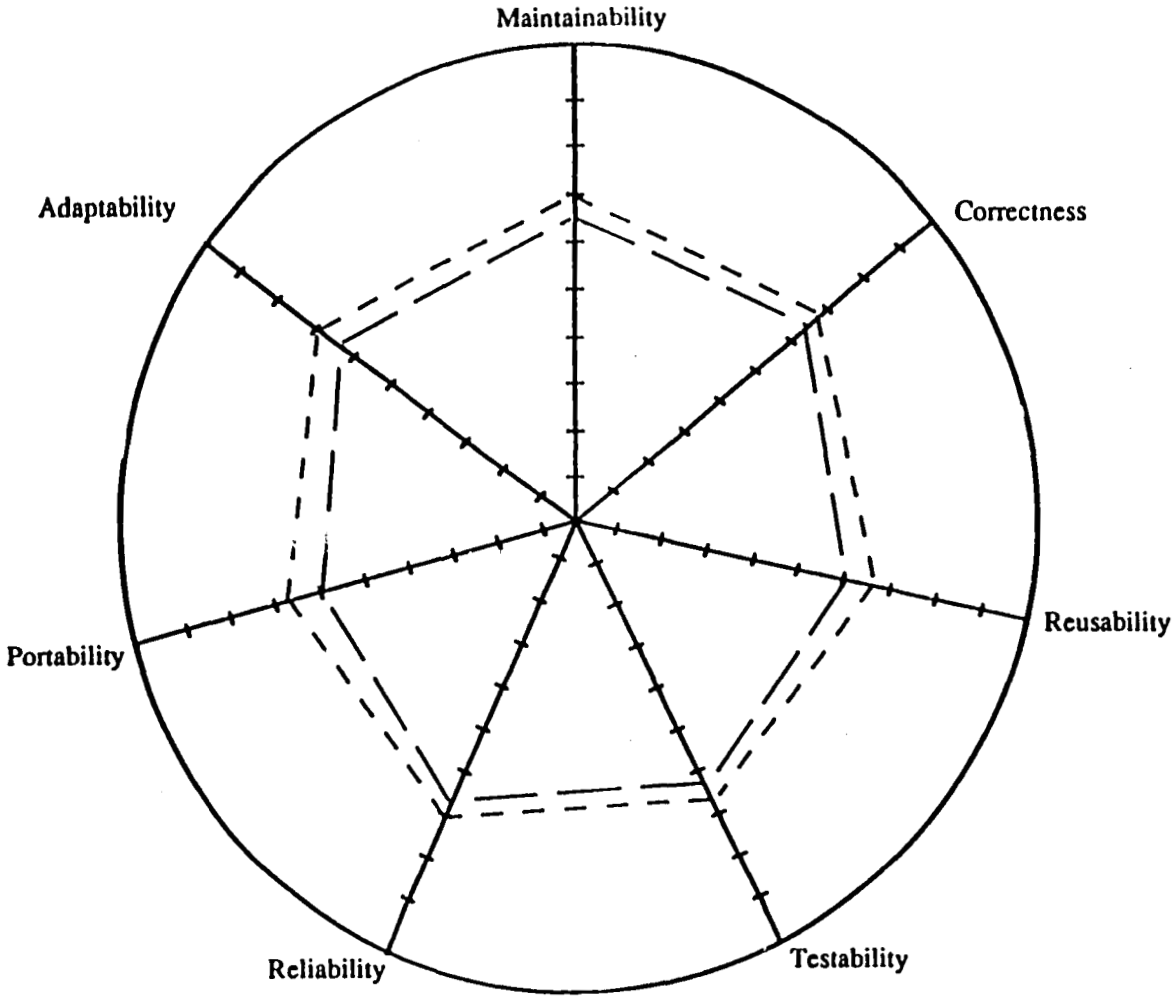
Methodology B – – – – – – –



KIVIAT GRAPH FOR PRINCIPLES

Methodology A — — — —

Methodology B – – – – – –



KIVIAT GRAPH FOR OBJECTIVES

Methodology A — — — —

Methodology B – – – – – –

RESULTS OF PROCEDURE APPLICATION

Assessing "Methodological" Adequacy

A: Stresses Objectives of Reliability and Correctness
Emphasizes Principle of Structured Programming

Methodology A was (and is) an "evolving methodology"

B: Stresses Objectives of Maintainability, Adaptability, Reliability, and Correctness
Emphasizes Principles of Modular Decomposition, Structured Programing and Concurrent Documentation

At the Objectives level, both "methodologies" support stated project objectives.

At the Principles level, both "methodologies" lack the enunciation of proper Principle usage.

No reference to desired Attributes is found

FUTURE RESEARCH

Applying the Evaluation Procedure to Software Quality Assurance

Predicting and/or assessing software quality necessitates a

- *Systematic* approach to
- *Assessing* product (or process) conformance with
- *Acceptance criteria* (standards and guidelines)

The Evaluation Procedure

- Currently supports a well-defined, systematic approach for evaluating software products, and
- Provides a rigorous framework for
  - Relating acceptance criteria based on attributes to software engineering principles and objectives, and
  - Defining acceptance levels based on measures reflecting the achievement of objectives, adherence to principles and realization of attributes.