Document Quality Indicators: A Framework for Assessing Documentation Adequacy

James D. Arthur and K. Todd Stevens

TR 90-60

Document Quality Indicators:
A Framework for Assessing Documentation Adequacy*

James D. Arthur
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061
(703) 231-7538

K. Todd Stevens
Wang Incorporated
Lowell, MA 01851
(508) 976-5088

Index Terms: Documentation Quality, Documentation Quality Indicators, Metrics, Factors, Quantifiers, Automated Assessment

Abstract

This paper presents case study results of a research effort funded by the Naval Surface Warfare Systems (NSWC) at Dahlgren, Virginia. The investigation focuses on assessing the adequacy of project documentation based on an identified taxonomic structure relating documentation characteristics. Previous research in this area has been limited to the study of isolated characteristics of documentation and English prose, without considering the collective contributions of such characteristics. The research described in this paper takes those characteristics, adds others, and establishes a well-defined approach to assessing the "adequacy" of software documentation. The identification of Document Quality Indicators (DQIs) provide the basis for the assessment procedure. DQIs are hierarchically defined in terms of document Qualities, Factors that refine Qualities and Quantifiers that provide for the measurement of Factors.

* Work supported by the U.S. Navy through the Systems Research Center under Basic Ordering Agreement N60921-83-G-A165 B0038.

1. Introduction and Background

The dual problem of high costs and low quality of software are general concerns in the software engineering community today. Generally, quality is sacrificed due to cost constraints, but the inverse occurs as well: costs are driven up in order to achieve higher levels of quality. The two problems are connected, in that deficiencies in the development process visit huge costs on the maintenance of many systems; it is claimed that more than one-half of the total life cycle costs are incurred during the maintenance phase (Boehm, 1976, Lientz, 1978). Contributing to this substantial burden are several factors:

- a severe shortage of *talent* in software engineering, especially lacking is the experience so vital for maintenance activities,
- a lack of complementary *methods* and *techniques* for guidance and direction of the maintenance support, and
- a scarcity of *tools* for supporting the maintenance activities so intrinsic to complex software.

Solutions to problems in this domain advanced with the acknowledgement of a *life cycle* perspective of the development process, which helped to focus attention on the root problems of early software/hardware systems. Aided by the visibility attributed to software maintenance, computer scientists have been able to explain the need for a disciplined development process, controlled through meticulous attention to configuration management and governed by an emerging set of fundamental principles (Arthur 1987, Murine, 1986). In practice, these principles are associated with objectives for a given software development activity and are assessable through associated attributes of the product (Arthur 1987).

Among these principles is *Concurrent Documentation*: the recording of requirements, design, specification and implementation decisions as they occur with the commitment to convey purpose,

content and clarity (Tausworthe, 1977). This principle is particularly important for the maintenance phase because good documentation may be the only means available to maintenance personnel to understand the internal details of the software and then intentions of the original developers. Unfortunately, as with other principles, the failure to observe the Concurrent Documentation principle is common in all application domains. Because Concurrent Documentation is so crucial to the software development process, and later, to the maintenance activity, the extent to which this principle is observed must be measurable and controlled. This leads to the problem of determining how one might evaluate the quality of documentation as used during the software development and maintenance activities.

The research presented in this paper does not attempt to prescribe another method by which one can guarantee that the principle of concurrent documentation is employed. It does suggest, however, one approach to *evaluating* the adequacy of documentation relative to the system that it purportedly describes. Intrinsic to this research effort has been the investigation of two fundamental questions:

- *What constitutes "adequate" documentation?*
- *How does one "measure" the adequacy documentation?*

Among the several findings coming from an investigation of the above questions, the following are viewed as key:

- the development of an Evaluation Taxonomy that relates document Qualities to Factors and document Factors to measurable Quantifiers, and
- the formulation of a general theory of documentation analysis based on the Document Quality Indicator (DQI) concept.

A discussion of how these findings enable one to effectively address and answer the above two fundamental questions is the focus of this paper.

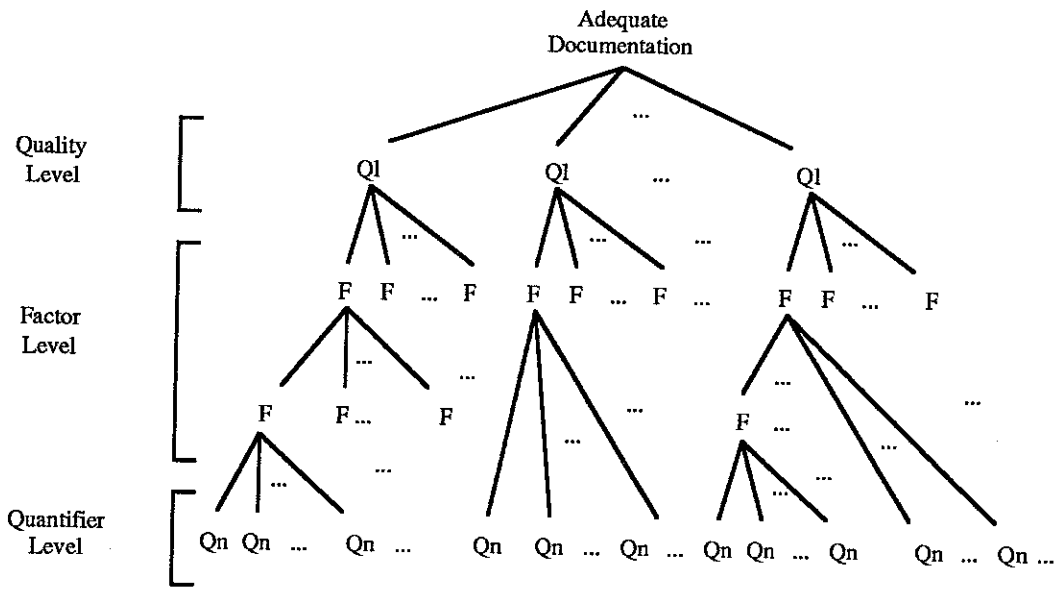

Figure 1

The Taxonomy Tree: A Hierarchical Structure

2. A Taxonomy for Identifying Document Quality Indicators

Seeking to identify what constitutes "adequate" documentation and to establish a way to measure that adequacy has led to a general taxonomy for the evaluation of documentation. During the process of identifying evaluative "characteristics" of documentation, it was noticed that some characteristics were descriptive, but too vague and subjective from a measurement perspective, whereas, others were quantifiable, but too narrow and questionably associated with documentation adequacy. Consequently, our research focused on a progressive refinement of the vague characteristics into the quantifiable ones. This approach produced a natural hierarchy of documentation characteristics with three "levels", viz., Qualities, Factors, and Quantifiers.

The hierarchical structure is best visualized as a *taxonomy tree* with the following top-down description. As illustrated in Figure 1, the nodes at the first level represent Qualities often associated with adequate documentation; the second level is populated with Factors, as they are related to the Qualities; finally, the third level represents measurable Quantifiers related to individual

Factors. As described in more detail later, the principal Qualities of adequate documentation are: Accuracy, Completeness, Usability and Expandability. Although these Qualities are not directly measurable, they can be further broken down into "smaller" and less abstract characteristics (or Factors) that do support the measurement of a particular Quality. For example, factual consistency and conceptual consistency are two Factors whose measurements are directly correlated with the perceived accuracy of a document. The evaluation taxonomy (from which Document Quality Indicators are determined) is a product of a decomposition and refinement process like the one implied above. The decomposition and refinement process is applied recursively until the most subordinate characteristics are tangible and directly measurable.

Implicit in this tree structure is the recognition that documentation adequacy can be determined through its Qualities, each of which must be assessed through Factors that are measured at the Quantifier level. Our search for *measurable surrogates* reflecting document qualities has led to the synthesis of Document Quality Indicators (DQIs). Intuitively, a DQI is a variable whose value can be determined through the direct analysis of document characteristics and whose evidential relationship to a document quality is undeniable. More precisely, a DQI is defined to be a triple whose elements are

- a Quality (of a document),
- a Factor (refining a Quality), and
- a Quantifier (measuring a Factor).

Clearly, the tree-structured taxonomy provides a natural basis for reasoning about documentation adequacy because it most naturally conveys the relationships among the contributors to document quality. Moreover, because each branch of the tree structure implies a reduction in the level of characteristic abstraction, each path from a Quality node to a Quantifier (or leaf) node *uniquely*describes a measurable quality indicator, i.e. a DQI.

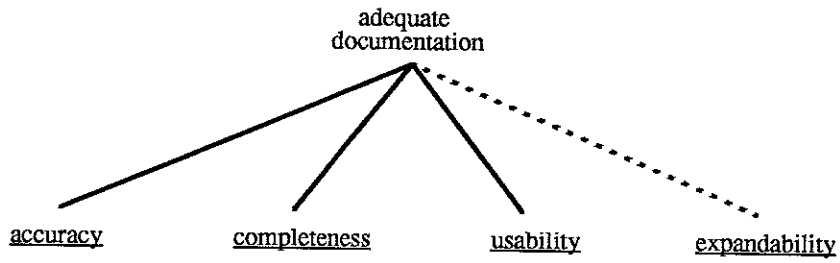

Figure 2
Decomposition into Qualities

3. Qualities of Adequate Documentation

A preliminary investigation and extensive literature review (U.S. ARMY, 1984, U.S. Air Force, 1987, Collofello, 1986, Horowitz, 1986a, Horowitz, 1986b, Murine 1986, Sneed 1985) reveals that the primary, high-level Qualities of good documentation to be: *Accuracy, Completeness, Usability and Expandability.* Qualities are the abstract characteristics of Adequate Documentation, yet, essential to its to its definition. As illustrated in Figure 2, Qualities are the nodes directly under and most closely tied to the even more abstract notion of Adequate Documentation. Although intangible, the identified Qualities most closely convey the meaning of Adequate Documentation; they are also the first of three elements that comprise the DQI triple. For the purpose of clarity, and to set the stage for the decomposition process leading to Factors of Qualities and Quantifers measuring Factors, the following paragraphs provide a brief description definition and description of all four qualities.

Accuracy

The common definition for "accuracy" is the freedom from mistake or error; a synonym is "correctness." Within the context of computer documentation, *Accuracy* can be defined as the consistency among the code and all documentation of the code, for all requirements. That is, accurate documentation should reflect the realized state of the system that it represents.

Inconsistencies among documents and the instantiated system can be introduced in several way. For example, errors can be introduced when system requirements are translated into design specifications, or when the design specifications are being interpreted and implemented as code. Effectively, the requirements can be correct and the code and/or development documentation incorrect. Alternatively, the product can be modified to correct an execution defect, but the corresponding development and/or requirements documentation might not be updated. As a result, some or all of the development documentation could be incorrect.

Completeness

The standard definition for "completeness" is the possession of all necessary parts, elements, or steps. For the purposes of computer documentation, a set of documentation is complete if all of the required information is present.

With respect to document Completeness, however, the major problem lies in determining what is required or needed. To determine what documentation is necessary for a computer system, one must first consider the computer system itself; computer systems vary significantly, so their documentation must necessarily vary. Their differences notwithstanding, standards do exist for every type of computer system; these standards define what is required for documentation completeness. Many such standards have already been established (ANSI 1974, IEE 1985, Poschmann 1984, U.S. Department of Commerce 1976) Hence, incorporating standards into the description, a more precise definition for *Completeness* in the context of document quality is: the existence of all documents required by a set of standards.

Usability

The dictionary definition for "usability" is the capability, convenience, or suitability of being employed. Relative to assessing documentation quality, *Usability* is more appropriately defined as the suitability of the documentation relative to the ease with which one can extract needed information. For example, part of assessing Usability is evaluating the Logical Traceability of

the documentation. That is, assessing the ease with which one can (a) locate an item or the presentation of a concept within a set of documents and/or (b) trace an item or the development of concept through different parts of the documentation.

Expandability

A general definition for "expandability" is the ability to increase an object's extent, number, volume, or scope. A synonym is "extensibility." The rationale for including expandability as a desirable Quality is to reflect concepts underlying document maintainability, and in particular, the ease with which the documentation can be added to and modified. In concert with the notion of document maintainability, a more precise definition for *Expandability* is: the capability of the documentation to be modified in reaction to changes in the system. This Quality is assessed through measures reflecting ease of modification.

Because the research effort discussed in the paper considers only pre-defined, static sets of documentation, the partitioning of Expandability into its constituent Factors and Quantifiers has not yet been completed. Nonetheless, Expandability is included here to complete the description of documentation Qualities relative to assessing the adequacy of documentation.

4. Factors of Accuracy

Accuracy, Completeness, Usability and Expandability are the four Qualities most directly related to adequate documentation. Because of their intangible nature, however, document Qualities elude direct measurement. As one moves from the concept of a document Quality to that of a Quality Factor, abstract connotations give way to more concrete perceptions. Based on the identification of Qualities of documentation and reflective of the tree-structured model, the following discussion focuses on Factors that refine Qualities. For presentation purpose, however, only Factors Accuracy will be considered. The identification and development of other documentation characteristics supporting the assessment of documentation adequacy follow similar development

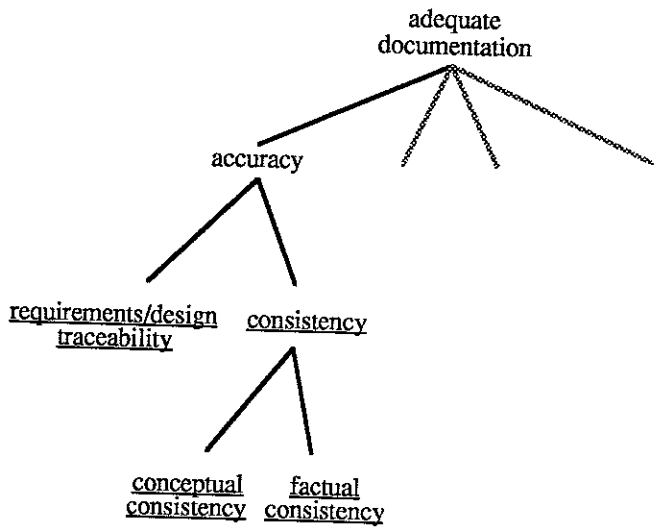

Figure 3
Decomposition to the Factor Level

paths. The authors refer the interested reader to (Stevens, 1988)for a more expansive description of that process and the resulting evaluation taxonomy.

In general, Factors can be defined as an essential part that contributes to the production of a result. Within the taxonomy framework, Factors are those more tangible characteristics that further refine each abstract document Quality. Factor are directly derived from Qualities, are less abstract than Qualities, and provide the second of three elements comprising the DQI triple. The remainder of this section describes the role that Factors play in defining a framework for assessing the quality of a document through DQIs.

To demonstrate that documentation exhibits the Quality of Accuracy, one must show that the documentation is consistent with the deployed system. Effectively, all documentation elements, from requirements to high- and low-level design documents, need to be *consistent* with the code and exhibit consistency among themselves. In order to assess consistency, pertinent information items must be linked in some manner and must be *traceable* from one documentation element to another. As illustrated in Figure 3 this traceability characteristic, along with two forms of consistency, are the Factors that support an assessment of documentation Accuracy. In the following sub-sections,

each Factor of Accuracy is discussed.

Requirements/Design Traceability

Requirements/Design Traceability is viewed as the ability to track individual system and design requirements to/from their corresponding manifestation in the source code. Requirements/Design Traceability provides the means for assessing the extent to which each level of documentation (from requirements to code) reflects the same set of requirements. Clearly, if the requirements and design documents state that requirement X must be met, yet, in the code we see a realization of Y, then one must question the accuracy of the documentation (or at least the implemented system). To achieve traceability, the requirements and design decisions must be enumerated as "atoms," with each requirement given as a single, indivisible entity. These atoms can then be traced through code and documentation, and assessed for consistency (Lamb 1978). In determining the extent to which requirements are traceable we must also recognize that requirements introduced because of design decisions must also be included in the assessment process.

As an example of requirements/design traceability, suppose that one system requirement states that specifications set forth in Standard "X1" must be met. Suppose further that among those specifications one finds the statement: "error messages must be logged in file: ERROR FILE." Not only do the high- and low-level designs have to reflect this requirement, the code must also *implement* it. The high- and low-level designs might, in fact, group the atoms together in the same general way that the system requirements do, such as by simply saying, "in accordance with Standard X1." Nonetheless, it is important for evaluation purposes that the requirements can be atomized because

- the requirements cannot be systematically assessed for consistency unless the requirements can be treated as atoms, and
- the requirements can break apart at the source code level, in that only some of them are germane to each section of the code.

The manner in which these requirements are actually assessed for traceability is described in Section 5.0.

Conceptual and Factual Consistency

Consistency is the agreement or harmony demonstrated among separate items. Relative to documentation, it is the agreement or concurrence of all information in the documentation. That is, the same idea must be expressed in a similar fashion or in a way that is not contradictory. Consistency has two facets: Conceptual and Factual. Conceptual Consistency means that an idea may be stated in different forms, but the forms must *convey* the same thought or notion. For example, one expects the physical representation of a stated requirement to change as it evolves from a description rooted in a requirements language, through a design language and finally culminating in an implementation. Nonetheless, the conceptual idea expressed in each of these forms must be consistent. Factual Consistency means that statements of value, logical relationships, and definitive structures must remain *invariant* irrespective of their location and repetition within the documentation. Many system requirements, for example, are stated as simple numeric or name facts, e.g. the system must be able to buffer a minimum of 10 requests. The design document would state the same numeric fact, and the code would explicitly reflect that lower bound.

Although in the next section we discuss only one approach to quantifying quality Factors, the authors would like to note here that consistency can (and should) be checked in two ways relative the software development life cycle, i.e. in a vertical direction and in a horizontal direction. Checking from the horizontal direction, all of the documents that are products produced *within* a particular life cycle phase must be consistent. Checking from the vertical direction, the products produced *across* different life cycle phases must exhibit consistency. Factual consistency must be checked in both directions because facts are presented a various places in the documentation at each stage of the development life cycle. Conceptual consistency, however, is primarily a concern in the vertical direction because concepts have the highest potential for change, through refinement, between stages of the life cycle.

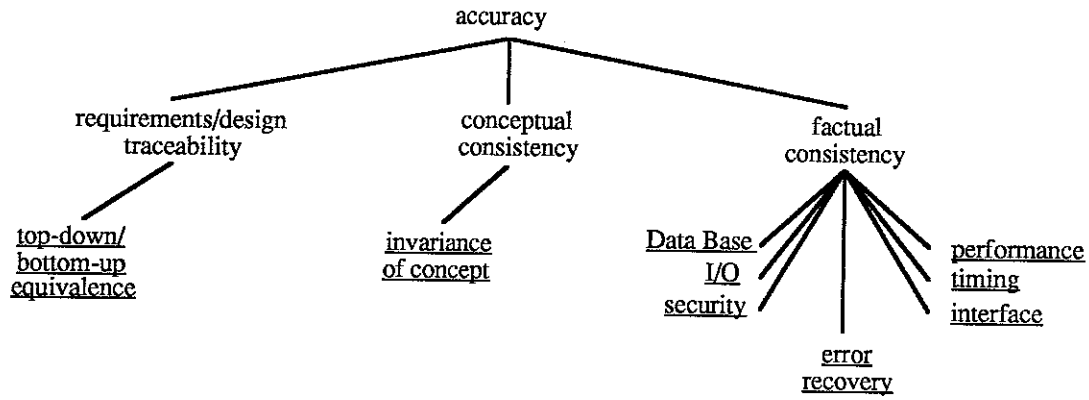

Figure 4
Quantifiers for Factors of Accuracy

The authors would also like to point out that even though a requirement is referenced in a consistent manner throughout software development life cycle phases and that even though traceability exists from the systems requirements specifications to the code, Documentation Accuracy is *not* necessarily assured. For example, some functionality might be included in the code for which no requirement exists. In reality one desires "equivalence" from a top-down (requirements to code) and bottom-up (code to requirements) assessment perspective. In the next section we illustrate how "top-down, bottom-up equivalence" completes one DQI triple and illustrates the DQI assessment perspective.

5. Quantifiers for Measuring Factors

Through the decomposition and refinement process, Factors are identified that support a more tangible basis for assessing documentation adequacy than do Qualities. Nonetheless, as conveyed by the examples given above, Factors of Quality are still missing the "concreteness" necessary to support measurement. Quantifiers, on the other hand, *are* measurable characteristics. Through their direct link to Factors, Quantifiers support Factor assessment, and ultimately, provide a basis for assessing documentation quality. In the taxonomy tree, Quantifiers are at the bottom, i.e. the leaves. Quantifiers particular to Factors of Accuracy are shown in Figure 4. Relative to the DQI concept, Quantifiers occupy the third position in each DQI triple. Hence, in conjunction with a related Factor and Quality, the addition of a Quantifier *uniquely* defines a DQI. As mentioned

in the previous section, the authors choose to restrict the following discussion to one Quantifier supporting the assessment of requirements/design traceability relative to documentation accuracy.

Requirements/Design Traceability: Top-down/Bottom-up Equivalence

Top-down/Bottom-up Equivalence is the equality between specified requirements and those implemented in the system source code. In order to meet a necessary-and-sufficient condition, equality must be assessed through a top-down and a bottom-up evaluation of the documentation and code. The requirements must be "sufficiently" met by the implementation, and all of the code must be "necessary" to meet the requirements, i.e. there is no superfluous code.

The top-down trace checks whether all of the requirements have been "sufficiently" met by the implementation. All of the requirements are enumerated as atoms, so that they can be uniquely identified and associated with separate features which are helping to meet or satisfy them. The separate features are then further divided, thereby dividing the requirements with which they are associated. The bottom-up trace checks whether all of the code modules are actually "necessary" or needed to meet a requirement.

This top-down/bottom-up tracking can best be accomplished by setting up evaluation matrices (see Figures5a and 5b). The matrix shown in Figure 5a relates requirements to design modules; the matrix in Figure 5b illustrates the relationships that exist between design modules and code modules. In the first matrix (Figure 5a), the requirements are enumerated across the top of the matrix and the design modules along the left side. For each requirement, an "X" is marked for the design module(s) that partially or completely fulfills that requirement. For the second matrix (Figure 5b), the design modules from the first matrix are enumerated across the top and the code modules along the left side. Similar to the first matrix, an "X" is marked for the code module(s) which partially or completely implements each design module.

Once the matrices are filled in, they are used to check that all of the requirements are met and that all of the code modules are necessary. If a requirement column in the first matrix does

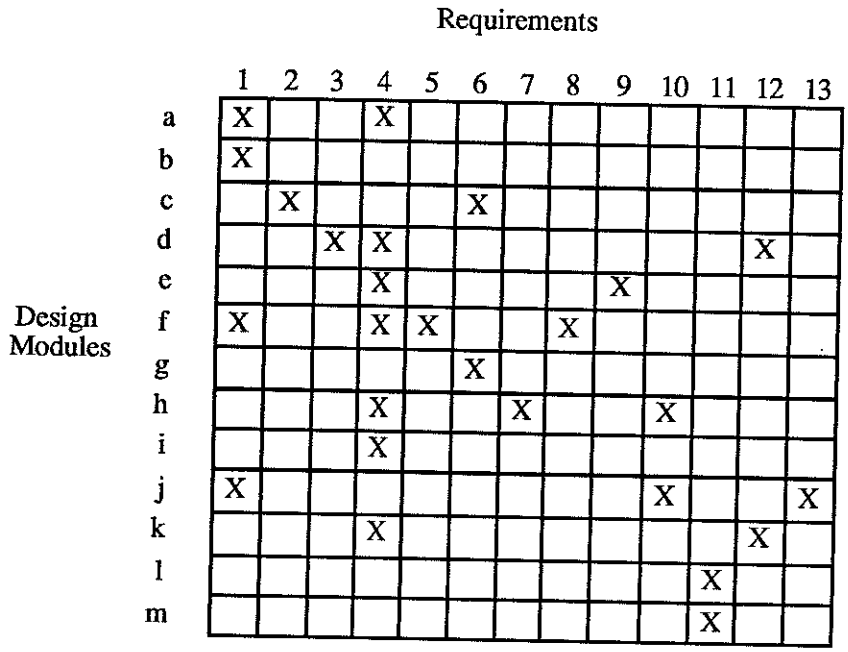

Figure 5a. Requirements/Design Matrix

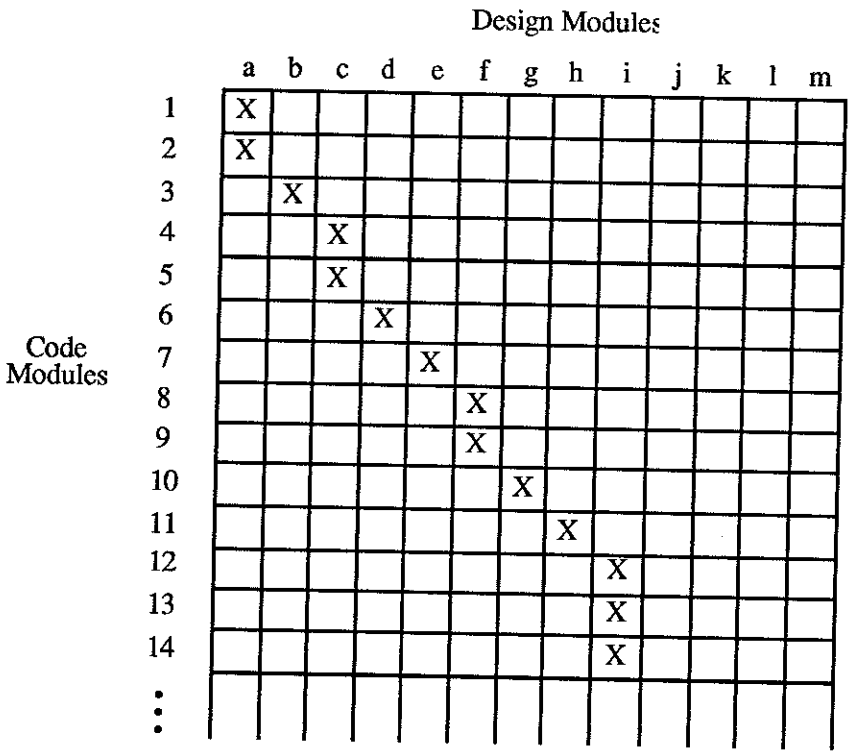

Figure 5b. Design/Code Matrix

not have at least one "X", this is evidence that no design module exists to implement it. Further, the requirements also depend on the design modules being implemented; each design in the second matrix which is not implemented in a code module fails to fulfill requirements. In this manner, the requirements can be checked to determine whether they are all *sufficiently* fulfilled. Next, the rows of the matrices need to be examined. Any row which does not have at least one "X" in it is not needed; it does not help to fulfill a requirement. This examination allows one to determine whether a design module is necessary to meet requirements.

6. Current Status

Although simplified, the use of matrices described in the previous section illustrates how one might check the documentation and code for "sufficiently meeting the requirements" and "necessary to meet the requirements", respectively. Using the one DQI, "accuracy relative to design and requirements traceability as reflected through top-down and bottom up equivalence," as an example, the above sections outline the relevance and importance of the DQI concept in document analysis. The complete taxonomy tree, illustrated in Figure 6, outlines the critical elements of a framework for assessing the adequacy of computer documentation, i.e. Document Quality Indicators. In particular, each path from the root node of the taxonomy tree to a leaf node *uniquely* defines a measurable DQI that supports the evaluation process. Through the relationships implied by each DQI triple, Quantifier *values* (provided by yet to be determined metric computation) can be related to specific document Qualities. In turn, the collective measure of the Qualities provide a means for assessing documentation adequacy. Currently, the investigative effort has led to the identification of

- four (4) major document Qualities,
- thirteen (13) Factors of Qualities, and
- thirty (30) unique Quantifiers, all combining to form
- thirty seven (37) document quality indicators (DQIs).

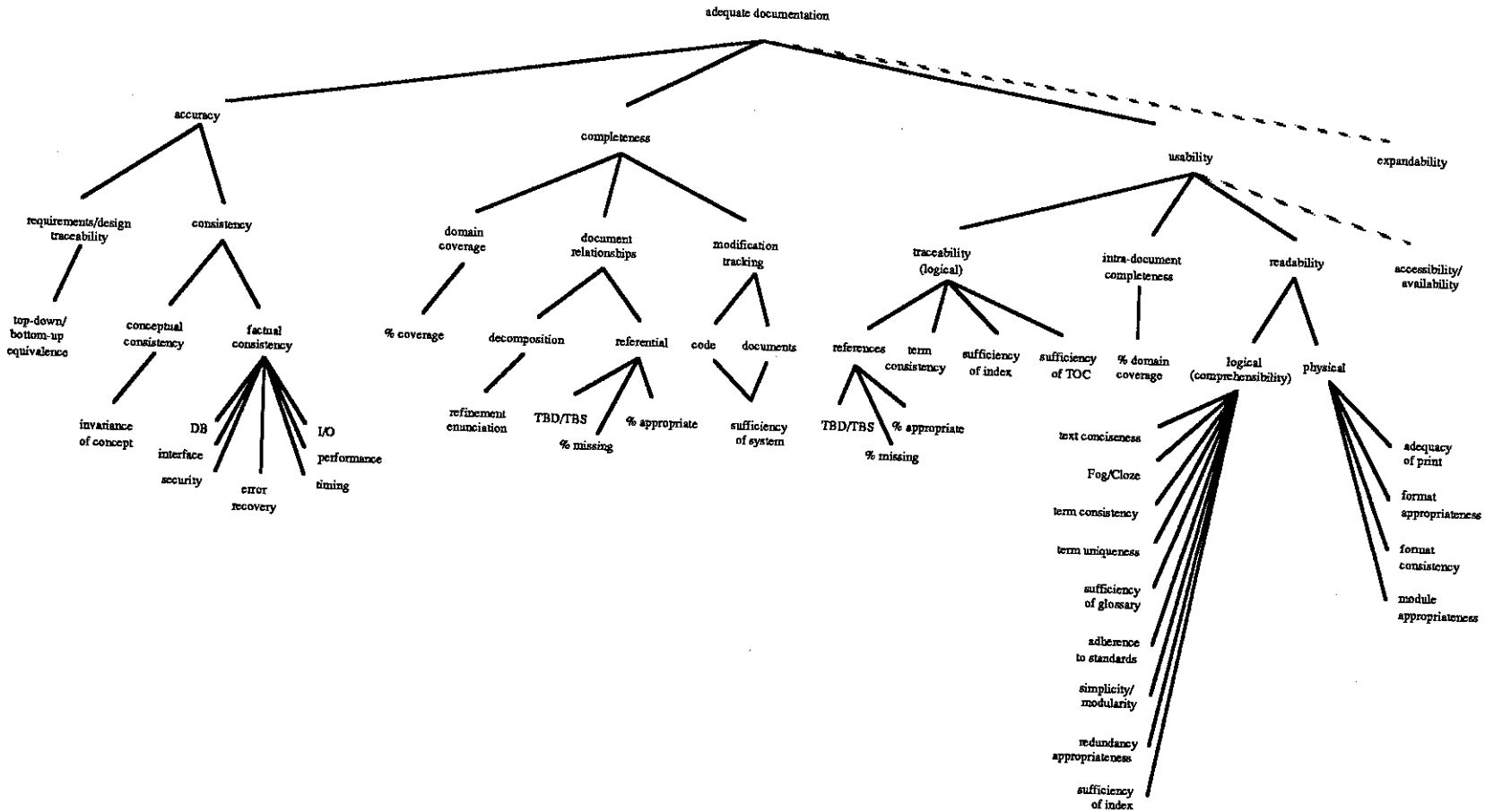

Figure 6

The General Evaluation Taxonomy Tree

7. Concluding Remarks

Motivating the research described in this paper is the recognition that system and project documentation are crucial for high quality software development and maintenance. Implicit in this recognition is that documentation must be accurate, complete and usable. Armed with this realization the authors have initiated a research effort that attempts to identify the crucial components of "adequate" documentation and to establish a meaningful basis for quantitatively measuring the contributions those components from both the individual and collective perspectives.

Although the evaluation taxonomy presented in this report represents a significant step toward assessing document adequacy, it remains only a framework. That framework, however, has been applied "in principle" to a set of documents used for maintenance purposes at the Naval Surface Warfare Center, Dahlgren, Virginia (Arthur, 1988). The results of that assessment are encouraging. Nonetheless, many issues still remain unresolved. The following paragraphs outline four future research directions that require additional investigation before a total process for document analysis can truly be realized.

The Quest for Additional Quantifiers: The first investigative effort involves a reassessment of the evaluation taxonomy focusing on substantiating the current DQIs and the formulation of additional ones. In particular, an emphasis needs to be placed on the identification of additional Quantifiers supporting the assessment of existing Quality/Factor pairs.

Metric Identification and Formulation: A second effort must address the identification and synthesis of metrics to support the computational process underlying document evaluation through DQIs. The findings presented in this report suggests several such metrics. Those metrics, however, are incomplete and primarily reflect the authors' intuition. Like the undeniable relationships on which DQIs are based, the proposed metrics must be succinct, well-defined and reflect a high level of reliability and validity with respect to the DQIs they are measuring. Clearly, such an investigation requires significant experimental studies validated through statistical confirmation.

An Automated Assessment Framework: An investigation is also needed which focuses on the identification of those metrics that can provide an automated framework for DQI assessment. Several of the DQIs identified in the current research suggest metrics that require raw data whose collection is manpower intensive and susceptible to bias. Ideally, one desires metrics whose data elements can be collected through an automated process predicated on objectivity. Recognizing that both subjective and objective metrics do exist and only some of either can be collected automatically, one possible research direction might focus on the identification of a subset of metrics that are (a) amenable to an automated evaluation framework, and (b) whose collective characteristics and evaluation implications represent those of the total set.

A Validation of the Assessment Procedure: Utilizing results from the above three efforts, the authors envision a final research effort aimed at validating the assessment procedure. Such an effort would require a collaborative investigation involving a sponsoring group that has at its disposal several sets of project documentation, knowledgeable people willing to provide an objective assessment of the documents, and an automated process for effecting the assessment process based on DQIs.

In summary, the research findings presented in this report provides a well-defined framework for the evaluation of documentation. The need for methods to evaluate documentation is readily apparent within the maintenance domain. The need becomes even more obvious when one considers assessing the quality of complete software systems through an evaluation of the developmental process as well as the developed product. As indicated above, however, difficult issues must be resolved before the long range goals of this investigative effort can be realized.

The authors would like to acknowledge the contributions of

- Dr. Richard E. Nance, whose initial work on Software Quality Indicators provided a basis for the DQI concepts, and
- Mr. Dave McConnell, Mr. Angel Martinez and Mr. Bob Barthelowe at NSWC (Dahlgren, VA), whose constant feedback had a significant impact in shaping and refining the underlying document characteristic taxonomy.

References

American National Standards Institute. "American National Standard for Guidelines for the Documentation of Digital Computer Programs," ANS-N413-1974, American Nuclear Society, 1974.

Arthur. J. D., Nance, R. E., K. T. Stevens, "Prospects for Automated Documentation Analysis in Support of Software Quality Assurance," Technical Report SRC-88-002, Systems Research Center, Virginia Tech, 1988.

Boehm, B. "Software Engineering," *IEEE Transactions on Computers*, C25 (December 1976), pp. 1226- 1241.

Collofello, J. S. and S. Bortman, "An Analysis of the Technical Information Necessary to Perform Effective Software Maintenance," *Proc. of the Fifth Annual Phoenix Conference on Computers and Communications*, March 1986, pp. 420-424.

Horowitz, E. and R. C. Williamson. "SODOS: A Software Documentation Support Environment -Its Definition," IEEE Transactions on Software Engineering, SE-12, No. 8 (August 1986), pp. 849-859.

Horowitz, E. and R. C. Williamson. "SODOS: A Software Documentation Support Environment -Its Use," *IEEE Transactions on Software Engineering*, SE-12, No. 11 (November 1986), pp. 1076-1087.

Institute of Electrical Engineers. *Guidelines for the Documentation of Software in Industrial Computer Systems*, The Institute of Electrical Engineers, 1985.

Lamb, S.S., et al., "SAMM: A Modeling Tool for Requirements and Design Specification." *Proceedings of COMPSAC 78*, Chicago, IL, 1978, pp. 48-53.

Lientz, B. "Issues in Software Maintenance," *ACM Computing Surveys*, 15, No. 3 (September 1983), pp. 271-278.

Murine, G. "Using Software Quality Metrics as a Tool for Independent Verification and Validation," *Proc. of the Fifth Annual Phoenix Conference on Computers and Communications*, March 1986, pp. 433-437.

Poschmann, A. W. *Standards and Procedures for Systems Documentation*, American Management Associations, 1984.

Sneed, H. M. and A. Merey. "Automated Software Quality Assurance," *IEEE Transactions on Software Engineering*, SE-11, No. 9 (September 1985), pp. 909-916.

Stevens, K. T., J. D. Arthur and R. E. Nance. "A Taxonomy for the Evaluation of Computer Documentation," Technical Report SRC-88-008, Systems Research Center, Virginia Tech, January 1988.

Tausworthe, R. C. *Standardized Development of Computer Software*, Prentice-Hall, 1977.

U.S. Air Force, "Acquisition Management: Software Maintainability - Evaluation Guide," AFOTEC Pamphlet 800-2, Vol. 3, March 1987.

U.S Army, *Software Quality Engineering Handbook*, Computer Systems Command, Ft. Belvoir VA, August 1984.

U.S. Department of Commerce. *Guidelines for Documentation of Computer Programs and Automated Data Systems*, Federal Information Processing Standards Publication 38, 15 February 1976.