JOURNAL OF AEROSPACE COMPUTING, INFORMATION, AND COMMUNICATION
Vol. 3, July 2006

Extracting Acyclic Dependency Models from Quality Standards for COTS Software Evaluation

A. Terry Morris*
NASA Langley Research Center, Hampton, Virginia 23681

and
Peter A. Beling†
University of Virginia, Charlottesville, Virginia 22903

Commercial off-the-shelf (COTS) software has become a crucial component of many of today's mission- and safety-critical products. As these products have proliferated internationally, global demand for software safety and performance quality has increased. Unfortunately, the current state-of-the-art in software assessment technology has not matured to the point where well-established and widely accepted schemes for evaluating COTS software quality have been presented. Without insight into the source code, COTS software users depend on a subjective assessment of attributes described by performance metrics, behavioral descriptions, and/or historical product data. Published techniques that attempt to formalize the above approach are often less than successful due to ill-defined and subjective metrics, and the naïve assumption of independence between attributes. Recently, software standards have been seen as a means of improving the evaluation process by providing an acknowledged criterion for software metric comparison. Given the stated issues, is it possible to develop a standards-based dependency model of well-defined attributes that does not depend on naïve assumptions? The answer to this question is yes. This paper describes a process that can be used to extract acyclic dependency models, if they exist, from various software product quality standards. Application of the extraction process will be used to reveal the attribute dependency model of the international software standard ISO/IEC 9126-1. Not only can this process be used for COTS software evaluation, but it can also be used by standards making bodies to clarify the reasoning, justifications and causal implications they employ when developing standards.

Nomenclature

AHP	=	analytical hierarchy process
CIS	=	conditional independence statement
CPD	=	conditional probability distribution
DAG	=	directed acyclic graph
COTS	=	Commercial off-the-shelf
$\boldsymbol{G}$	=	a graph of topological dependence structure
$I(.,.\vert .)$	=	a conditional independence rule

Received 26 July 2005; revision received 14 April 2006; accepted for publication 12 May 2006.

* Research Engineer, Safety-Critical Avionics Systems Branch, Mail Stop 130, AIAA Lifetime Associate Fellow.
† Associate Professor, Department of Systems Engineering, Thornton Hall.

ISO/IEC	=	International Organization of Standardization/International Electrotechnical Commission
JPD	=	joint probability distribution
OTSO	=	Off-the-Shelf-Option
$p(x\|\pi_x)$	=	a conditional probability distribution
π_x	=	parents of node *x*
SEI	=	Software Engineering Institute (SEI)
WAS	=	weighted average sum
$\boldsymbol{X}, \boldsymbol{Y}$	=	sets of variables
x, y	=	instantiated variables

I. Introduction

THE international proliferation of COTS software products has increased the global demand for software safety and performance quality. Unfortunately, the current state-of-the-art in software assessment technology has not matured to the point where well-established and widely accepted schemes for evaluating COTS software have been presented. Many of the published evaluation techniques for COTS software are ad-hoc. For these, the structure and definitions of the attributes are generally not objective in that they usually depend on the user. These techniques may also rely on naïve assumptions of independence between attributes. Since source code is generally not included with most COTS software products, any evaluation technique using software metrics must include formalisms for handling the intrinsic uncertainty. There is currently no state-of-the-art criterion for software metric comparison that incorporates attribute dependency relationships and the uncertainties involved with COTS software evaluation. There is a need then to establish a criterion by which to model attribute dependencies for COTS software evaluation.

Recently, software quality standards have been seen as a means of improving the evaluation process by providing an acknowledged criterion for software metric comparison. Generally, these software standards and the software metrics within them are not described with precise mathematical rigor in a manner that can be easily used. This paper presents an approach for extracting a standards-based dependency model of well-established attributes for COTS software evaluation. That is, given a software product quality standard, the approach extracts a dependency model that represents the causal relationships between the software attributes. This dependency model can then be used for probabilistic evaluation of COTS software products given historical COTS software data. Background information related to COTS software evaluations, software quality standards and dependency models are described in the next section. Section III provides an overview of the dependency model extraction process while section IV applies the extraction process to the international standard ISO/IEC 9126-1.[1]

II. Background

The dependency model extraction process (to be described in the next section) was one of a number of contributions developed within an overall methodology for COTS software scoring.[2] The holistic methodology incorporated many subject areas related to complex systems modeling, requirements analysis, software evaluations, and dependency networks. For the sake of brevity, only the areas of primary interest to the extraction process will be discussed. These areas include the state of COTS software evaluations, the role of software quality standards, and dependency models.

A. COTS Software Evaluation

COTS software products are products purchased from commercial vendors where the vendors control the product's development and future evolution. Purchasers of COTS software products often do not know the internals of the product because they are not given access to the source code.[3] In this sense, COTS software is viewed as a *black-box* item. Over the past decade, there has been an expanding effort by business and government to incorporate more pre-existing software into their systems. Some federal agencies have even gone so far as to establish policy requiring software procurers to justify why they are not using COTS products.[4,5] The rationale for utilizing COTS is primarily to lower development costs and time, to reduce maintenance effort, and to take advantage of advancements in technology. This increased dependence on COTS software has introduced substantial risks to software procurers, particularly those involved with safety or mission-critical systems. The risks stem from the fact that source code is generally not available and there is no control over the evolution of the product.[6] In 2001, it has been estimated that 99 percent of all executing computer instructions come from COTS products.[7] Thus, COTS software use is driven by economic necessity. Despite this necessity, the benefits, costs, and other risks of COTS software should be carefully weighed against other options. Several organizations have approaches for managing these risks. One such organization, the American Institute for Aeronautics and Astronautics (AIAA) is producing a national guidebook that outlines the benefits and risks of COTS software.[8] The guidebook elucidates how these risks can be mitigated. The lack of visibility into source code introduces worse case uncertainty when analyzing whether or not to accept COTS software. In what ways can this uncertainty be minimized and/or mitigated? In various safety- and mission-critical systems, COTS software has been incorporated with the introduction of protectors and/or wrappers.[9] For aerospace systems such as the Space Shuttle, COTS software can be included as long as its behavior can be analyzed using standard risk assessment processes.[10] These procedures require some form of visibility into the source code but do not necessarily imply modification of it. For other types of mission-critical systems, the acceptance of COTS software is based on evaluation and testing to assess the risks involved.

Software is a complex, intellectual product that commonly contains undesirable defects or faults. Checking the software's behavior against specified software requirements can generally identify the faults. Identification of software faults has traditionally been assessed via testing. Exhaustive testing, however, is intractable since programs generally have a combinatorial explosion of input states and paths.[11] Alternatively, projects usually try for coverage, that is, test all statements in the software's code at least once. Though intuitive, coverage testing really does not accomplish much since many software programs can have many states. The best result from coverage testing is that "warm fuzzy" feeling that guarantees nothing in the context of scientific certainty. The introduction and proliferation of COTS software products dampens the degree of reliability that is reliable by software evaluation since exhaustive and coverage testing cannot be performed due to lack of visibility of the source code. In this scenario, evaluating techniques that quantify the degrees of acceptance of a product must be used based on data that represents actual behavior of the product. These data can be garnered from software and qualification testing, vendor design specifications, end user product testing, customer experiences with the product, etc.

An underlying issue faced when selecting a software evaluation technique is determining software metrics, that is, the definition and treatment of attributes. Attributes are the most basic building blocks used for any evaluation. Fenton[12] provides a measurement framework by which a product can be evaluated. The framework is designed for users involved in the implementation of a metrics program from a developer's perspective. Fixed quality model approaches[13,14] identify attributes customized for a particular domain. Though practical, these approaches are not general. Constructive quality model approaches[1,15–17] allow maximum flexibility for attribute development; however, time and expertise are required for attribute customization. The most widespread of these approaches is the international standard ISO/IEC 9126,[1,15] which discusses quality, functional and non-functional attributes. Despite the conciseness of the standard, Briand[18] points out that architectural attributes must also be considered.

Generic evaluation issues relevant to this research are addressed in Briand's approach,[18] that describes the evaluation process as an operations research optimization problem, in the software evaluation tool by Cardenas-Garcia[19] which addresses the issue of evident uncertainty always present in evaluation, by Carney[20] from Carnegie Mellon University's Software Engineering Institute (SEI) who demonstrates that evaluation cannot be performed without knowing the context with which it is used, and by Dick[21] who makes the case that purely objective evaluation alone is not sufficient for evaluation of software. Jeanrenaud[22] describes a process by which software characteristics can be decomposed to a level where measurable metrics can be defined. Moreover, this research also suggests the importance of incorporating a strategy that considers risk in choosing the rigor of the evaluation technique applied. An interesting evaluation technique described by Kontio[23] involves a systematic process used to evaluate map applications and web browsers. This technique, entitled Off-the-Shelf-Option (OTSO), is used to select pre-packaged COTS and off-the-shelf components. The decomposition of criteria used for OTSO is based on the Goal-Question-Metric method and Saaty's criteria decomposition approach.[24] Analysis of the approach suggests that the use of the weighted average sum (WAS) and the analytical hierarchy process (AHP) aggregation techniques gave different final results and that WAS gave less insight into the evaluation process. This implies that the choice of aggregation technique should not be arbitrary and that it should be justified and validated. Another technique termed Iusware, developed by Morisio,[25] is a very promising methodology used for software evaluation and acceptance. The methodology, founded in multi-criteria decision theory, attempts to cover difference measurement scales and diverse aggregation techniques in the process of developing a rational decision making process. A criticism of this approach is the lack of consistency with regard to the aggregation technique employed. The client is inundated with a plethora of aggregation techniques from which to choose with no real guidance as to the benefit/detriments of each. Other evaluation approaches[26,27] appear to have the same common problem of specifying a justifiable and consistent aggregation technique whether they are related to safety-critical systems, cost analysis, or operating systems.

The approach taken in this research overcomes many of the detriments of the techniques described. It succeeds by providing a state-of-the-art criterion for software evaluation that uses clearly defined attributes and sub-attributes based on software product quality standards. The overall goal of the paper is the development of a process that extracts dependency models which represent the causal relationships of a software quality standard. This dependency model can then serve as a coherent, justifiable and consistent aggregation technique for COTS software evaluations.

B. The Role of Software Standards

Software standards provide a criterion or an acknowledged measure of comparison for quantitative or qualitative value for software. They also reduce confusion in fields such as software technology. Theoretically, if the software was developed according to a provable formal model, there is an objective structure by which to provide a criterion. However, since formal methods (theorem proofs) are not likely to be applied to COTS software, it is generally accepted that there is not an objective measure of truth. Software standards provide a means of establishing a criterion. They are usually developed by consensus and then adopted by local, national and/or international bodies. There are, in essence, two general approaches used to ensure software quality. The first involves assuring the process by which the software product is developed. The second involves evaluating the quality of the end product. The research in this paper will be restricted to software standards that describe a set of quality characteristics of the end product and can serve as a basis for quantitative evaluation. The most pronounced software quality standard used for this purpose is ISO/IEC 9126 started in 1985 and published in 1991.[15] The revision of this international standard, ISO/IEC 9126-1,[1] released in 2001 will serve as the primary standard used in this research. The ISO/IEC 9126-1 standard, serving as an example in this research, is by no means restrictive since other comparable standards that contain quality attribute for quantitative evaluation can be used.

A software product quality standard, such as ISO/IEC 9126-1, can be viewed as a knowledge base of software attributes, sub-attributes and their relationships developed by expert consensus. In the case of ISO/IEC 9126-1, at least 75% of the national bodies that voted were required for approval of the standard. The experts in this standard provided six quality characteristics (attributes) and guidelines for their use. Additionally, the standard supports software product evaluation by providing a quality model framework that explains the relationships between different approaches to quality (see Fig. 1). Clear definitions of attributes and supporting sub-attributes are also provided (see Fig. 2). Information in the standard describes the causal relationships between the attributes and sub-attributes. Quality-in-use attributes (attributes that represent the user's view of quality when the software product is used in a specific environment and a specified context of use) are also included (see Fig. 3).

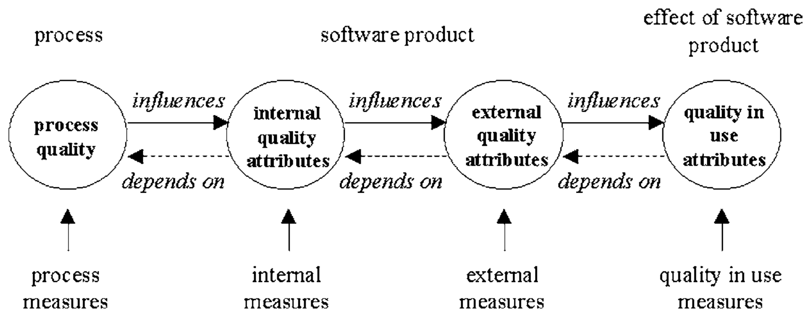


Fig. 1 ISO/IEC 9126-1 software quality framework.[1]

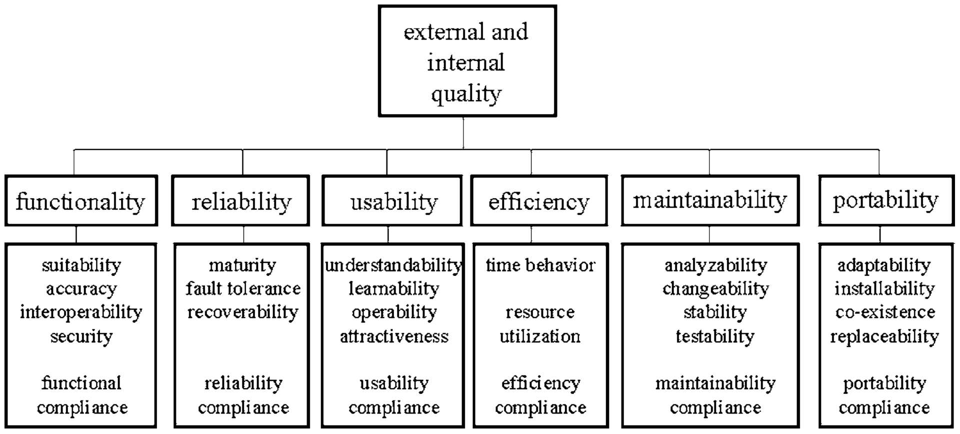


Fig. 2 ISO/IEC 9126-1 external and internal quality attributes.[1]

C. Dependency Models

Dependency is a statement about a set of variables. Saying that variables $\boldsymbol{A}$ and $\boldsymbol{B}$ are dependent on each other means that if the value of **A** is known, then information has been provided (to some extent) about the value of variable $\boldsymbol{B}$. Dependency can be conditional or unconditional, unidirectional or bidirectional, and functional or causal. The type of dependency of interest in this research is causal pair wise conditional dependency. Dependency models, as used in this paper, are sets of dependency statements. Formally, a *dependency model*[28] is a pair $\boldsymbol{M} = (\boldsymbol{U}, I)$, where $\boldsymbol{U}$ is a finite set of elements or variables, and $I(.,.|.)$ is a rule that assigns truth values to a three place predicate whose arguments are disjoint subsets of $\boldsymbol{U}$. The interpretation of the conditional independence assertion $I(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z})$ is that having observed $\boldsymbol{Z}$, no additional information about $\boldsymbol{X}$ could be obtained by also observing $\boldsymbol{Y}$. In a probabilistic model, $I(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z})$ holds if and only if

$$\boldsymbol{P}(\boldsymbol{x}|z, \boldsymbol{y}) = \boldsymbol{P}(\boldsymbol{x}|z) \text{ whenever } \boldsymbol{P}(z|\boldsymbol{y}) > 0 \tag{1}$$

for every instantiation x, y, and z of the sets of variables $\boldsymbol{X}, \boldsymbol{Y}$, and $\boldsymbol{Z}$.

A graphical representation of a dependency model $\boldsymbol{M} = (\boldsymbol{U}, I)$ is a direct correspondence between the elements in $\boldsymbol{U}$ and the set of nodes in a given graph, $\boldsymbol{G}$, such that the topology of $\boldsymbol{G}$ reflects the independence assertions of I. There are different kinds of graphical models. The most common are undirected graphs (Markov networks) and directed graphs (Bayesian networks). Each one has its own merits and shortcomings, but neither of these two representations has more expressive power than the other.[29] These graphical models are knowledge representation tools used by an increasing number of scientists and researchers. The reason for the extended success of graphical models is their capacity to represent complexity and to handle independence relationships, which has proved crucial for the storage of information. The graphical models used in this paper will be restricted to Bayesian networks.

Graphical models that represent directed dependencies are known as Bayesian networks and they result in a powerful knowledge representation formalism based on probability theory. Bayesian networks are graphical models where the nodes represent random variables, the arcs signify the existence of direct causal influences between the variables, and the strengths of these influences are expressed by forward conditional probabilities.[28] Formally, a

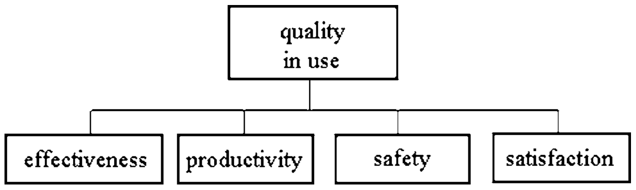


Fig. 3 ISO/IEC 9126-1 quality-in-use attributes.[1]

Bayesian network is a pair, $\mathbf{B} = (\boldsymbol{G}, \boldsymbol{P})$, defined by a set of variables $\boldsymbol{X} = (X_1, \ldots, X_n)$, where $\boldsymbol{G}$ is a directed acyclic graph (DAG) defining a model $\boldsymbol{M}$ of conditional dependencies among the elements of $\boldsymbol{X}$,

$$\boldsymbol{P} = (p(x_1|\pi_1), \ldots, p(x_n|\pi_n)) \tag{2}$$

is a set of n conditional probability distributions (CPDs), one for each variable, and π_i is the set of parents of node $\boldsymbol{X}_i$ in $\boldsymbol{G}$. The set $\boldsymbol{P}$ encodes the conditional independence assumptions of $\boldsymbol{G}$ to induce a factorization of the joint probability distribution (JPD) as

$$p(x) = \prod_{i=1}^{n} p(x_i|\pi_i). \tag{3}$$

When the random variables are discreet, as used in this research, the types of distribution applied to each variable will be multinomial, thereby describing a multinomial Bayesian network. In multinomial Bayesian networks, all variables in $\boldsymbol{X}$ are discrete, that is, each variable has a finite set of possible values. An advantage of Bayesian networks is its natural perception of causal influences thus making it an unambiguous representation of dependency.[30] This is useful for the COTS evaluation problem in that it allows for the explicit identification of influences between attributes of each software product. Moreover, the Bayesian network's requirement of strict positivity allows it to serve as an inference instrument for logical and functional dependencies. Furthermore, its ability to quantify the influences with local, conceptually meaningful parameters allows it to serve as a globally consistent knowledge base. In this way, Bayesian networks are natural tools for dealing with uncertainty and complexity. Characterizations of these networks generally involve determining whether the structure of the model is known and whether the data is complete. Bayesian networks with unknown structure and incomplete data are known to be NP-complete.[31] Bayesian models (when the structure is unknown) may be constructed from prior knowledge, conditional independence statements (CIS), data, or some hybrid of the above. Prior knowledge may include distinct knowledge of a problem domain as well as input from experts. As used in this paper, Bayesian networks, $\mathbf{B} = (\boldsymbol{G}, \boldsymbol{P})$, are graphical and mathematical models that represent the dependency and probabilistic structure between a set of variables. They can be used intuitively as a knowledge base for a set of variables where $\boldsymbol{G}$ (dependency structure) is the structural component and $\boldsymbol{P}$ (the probabilistic structure) is the more quantitative component. The dependency structure ($\boldsymbol{G}$) of a Bayesian network is usually extracted from domain experts. The term *probability model* refers to a complete specification of the JPD over a set of variables. Therefore, the terms probability model and JPD are used interchangeably. The JPD contains structural as well as quantitative information about the relationships among the variables. We will use the term *dependency model* to refer only to the causal structure of the relationships among a set of variables.

D. An Opportunity to Extract Dependency Models from Software Quality Standards

Since software product quality standards can serve as knowledge bases of software attributes and their relationships and Bayesian networks are graphical representations of the dependency structure between a set of variables, there is considerable overlap between the type of information in standards and the dependency structure used in a Bayesian network. Given this overlap, there is an opportunity to extract Bayesian network attribute dependency structure (topology) from software product quality standards.

III. Overview of Dependency Model Extraction Process

The dependency model extraction process describes the transformation process that maps the dependency structure of a set of attributes defined in a software product quality standard to a graphical representation of that structure as a dependency model, specifically a Bayesian network. An overview of the Bayesian network extraction process is shown in Fig. 4. This six step process starts with a software product quality standard. The process requires the identification of the relational/causal phrases in the standard that define attribute relationships. If possible, the process requires the translation of each causal relation into a CIS to be included in a causal list. After the list is reconciled (to prevent duplications and to include missed relationships elsewhere in the standard), qualitative forms of local CPDs are developed and then aggregated to form the qualitative JPD structure. This qualitative JPD is then represented graphically and then checked for cycles using a multilevel representation. If no cycles exist, then the output of the process reveals a qualitative DAG that represents the dependency model of the attributes in the software product quality standard. Quantitative instantiation of the dependency model is outside the scope of this paper.

Bayesian Network Structure Extraction

Input: A Software Quality Standard containing relational/causal phrases that describe attribute relationships
Output: A qualitative DAG that encodes the attribute/sub-attribute dependencies in the software standard

Step 1. Identify software quality standard relational/causal phrases that define attribute/sub-attribute relationships.
Step 2. Where possible, translate each causal relation into a conditional independence statement (CIS) and gather to form causal list.
Step 3. Reconcile the CISs to include missed causal relations located elsewhere in the software standard.
Step 4. Using the causal list, generate qualitative forms of local CPDs, then aggregate to form the qualitative JPD structure.
Step 5. Represent the JPD graphically allowing only one node for each attribute/sub-attribute to form the dependency model.
Step 6. Check for cycles in the dependency model; if cycles exist, model is not a DAG (thus, discard); if cycles do not exist, then model is a DAG (thus, DAG represents software standard dependencies).

Fig. 4 Bayesian network structure extraction process.

IV. Application: Dependency Model Extraction for the ISO/IEC 9126-1 Quality Standard

The best way to demonstrate the Bayesian network structure extraction process is by way of an example. The process in Fig. 4 will be used to capture and extract attribute dependencies within the ISO/IEC 9126-1[1] software product quality standard and will represent these causal dependencies graphically using a Bayesian network. Each step in the extraction process follows:

A. Input

The ISO/IEC 9126-1 Software Product Quality Standard will be used as input to the extraction process.

B. Step 1

In the first step, software quality standard relational/causal phrases that define attribute/sub-attribute relationships will be identified. The causal phrases that describe attribute/sub-attribute relations in ISO/IEC 9126-1 are primarily found in clause 6 (quality model for external and internal quality) and clause 7 (quality model for quality-in-use) of the standard[1]. Each clause describes several attributes (six for external/internal attributes and four for quality-in-use attributes) and the sub-attributes that categorize each attribute (if any). The text that describes each attribute/sub-attribute also describes the causal influences between the attributes. Many of these causal relations are found under the 'Notes' listed after each attribute/sub-attribute definition. Causal relations can generally be identified by scanning for the following words: 'affects,' 'will affect,' 'may affect,' 'influences,' 'may be influenced,' etc. When no causal relation can be identified, the attribute is considered independent. The following fifteen phrases were identified in ISO/IEC 9126-1 (referenced according to their location in the standard):

1) Clause 6 $\rightarrow$ external/internal quality is characterized by 6 attributes (functionality, reliability, usability, efficiency, maintainability, and portability) that have minimal overlap (clause C.3)
2) Clause 6.1 $\rightarrow$ Functionality is influenced by suitability, accuracy, interoperability, security, and functional compliance
3) Clause 6.1 Note 3 $\rightarrow$ suitability also affects operability
4) Clause 6.2 $\rightarrow$ reliability is influenced by maturity, fault tolerance, recoverability, and reliability compliance
5) Clause 6.3 $\rightarrow$ usability is influenced by understandability, learnability, operability, attractiveness, and usability compliance
6) Clause 6.3 Note 1 $\rightarrow$ some aspects of functionality, reliability, and efficiency will also affect usability

7) Clause 6.3.3 Note 1 $\rightarrow$ aspects of suitability, changeability, adaptability, and installability may affect operability
8) Clause 6.4 $\rightarrow$ efficiency is influenced by time behavior, resource utilization, and efficiency compliance
9) Clause 6.5 $\rightarrow$ maintainability is influenced by analyzability, changeability, stability, testability, and maintainability compliance
10) Clause 6.5.2 Note 2 $\rightarrow$ changeability may affect operability
11) Clause 6.6 $\rightarrow$ portability is influenced by adaptability, installability, co-existence, replaceability, and portability compliance
12) Clause 6.6.1 Note 2 $\rightarrow$ adaptability may affect operability
13) Clause 6.6.2 Note $\rightarrow$ installability can affect suitability and operability
14) Clause 7 $\rightarrow$ quality-in-use is categorized by 4 attributes (effectiveness, productivity, safety, and satisfaction)
15) Clause 7.1.3 $\rightarrow$ (safety) risks are usually a result of deficiencies in the functionality, reliability, usability or maintenance

C. Step 2

Where possible, each causal relation should be translated into a conditional independence statement and gathered into a causal list. When $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent given $\boldsymbol{Z}$, we write $I(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z})$. The statement $I(\boldsymbol{X}, \boldsymbol{Y}|\boldsymbol{Z})$ is referred to as a CIS. This statement conveys the idea that once $\boldsymbol{Z}$ is known, knowing $\boldsymbol{Y}$ can no longer influence the probability of $\boldsymbol{X}$. A good way to translate a causal relation to a CIS is by the use of the form

$$I(X_i, \boldsymbol{\phi}|\pi_i), \quad i = 1, \ldots, n \tag{4}$$

where $\boldsymbol{X} = (X_1, \ldots, X_n)$ represents the set of attribute nodes and π_i represents the set of parents that influence $\boldsymbol{X}_i$. Tables 1 (external/internal nodes) and 2 (quality-in-use nodes) display the CIS for each attribute/sub-attribute extracted from the standard.[1]

D. Step 3

At this point, the CISs should be reconciled to include missed causal relations located elsewhere in the software standard. Software standards are rarely written in a linear fashion. Because of this, causal relations associated with an attribute may appear with its parents (ancestors) or with its children (descendents) which may be located elsewhere in the standard. Two causal relations were updated after extracting and associating additional cause and effect relationships in the standard. One of the two relations dealt with the sub-attribute *suitability*. Initially, *suitability* (*suit*) was found to be independent, that is, $I(suit, \boldsymbol{\phi}|\boldsymbol{\phi})$. After further investigation, it was found that "*installability can affect suitability.*" The reconciliation of these relations led to the modified CIS, $I(suit, \boldsymbol{\phi}|inst)$.

The second reconciled sub-attribute was *operability*. This reconciliation was due to ambiguous wording in the standard. In particular, the ambiguous wording led to multiple meanings forming at least two competing interpretations of causal structure. The ambiguity revolved around the interpretations of clause 6 (external/internal quality is characterized by 6 attributes—*Functionality, Reliability, Usability, Efficiency, Maintainability*, and *Portability*—that have minimal overlap), clause 6.3 Note 1 (some aspects of *Functionality, Reliability*, and *Efficiency* will also affect *Usability*) and clause 6.3.3 Note 1 (aspects of *suitability, changeability, adaptability*, and *installability* may affect *operability*).

Interpretation #1 (reconcile ambiguity by adhering to major overarching causal phrases in case of conflict)

Using this interpretation, the major overarching causal phrase is stated in clause 6 (external/internal quality is characterized by 6 attributes—*Functionality, Reliability, Usability, Efficiency, Maintainability*, and *Portability*—that have minimal overlap). Clause 6.3 Note 1 is non-specific and quite ambiguous because it does not specify which aspects of *Functionality, Reliability*, and *Efficiency* will affect *Usability*. Without further clarification, this causal phrase can be interpreted in at least two different ways. First, some aspects (some sub-attributes) of these attributes affect the attribute *Usability.* Or, second, these attributes affect the attribute *Usability* directly. Note that most of the other causal phrases deal with how sub-attributes affect or influence other attributes or sub-attributes and not how attributes affect other attributes. Clause 6 appears to be the only other causal phrase that states how attributes relate

Table 1 ISO/IEC 9126-1 causal list and CPDs for external/internal attributes.

Node (symbol)	CIS	CPD
Functionality (F)	$I(F, \phi \mid suit, acc, intrp, sec, fc)$	$p(F \mid suit, acc, intrp, sec, fc)$
Suitability (suit)	$I(suit, \boldsymbol{\phi} \mid \boldsymbol{\phi}) \rightarrow I(suit, \boldsymbol{\phi} \mid inst)$	$p(suit \mid inst)$
Accuracy (acc)	$I(acc, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(acc)$
Interoperability (intrp)	$I(intrp, \phi \mid \phi)$	$p(intrp)$
Security (sec)	$I(sec, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(sec)$
F compliance (fc)	$I(fc, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(fc)$
Reliability (R)	$I(R, \boldsymbol{\phi} \mid mat, ft, rec, rc)$	$p(R \mid mat, ft, rec, rc)$
Maturity (mat)	$I(mat, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(mat)$
Fault tolerance (ft)	$I(ft, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(ft)$
Recoverability (rec)	$I(rec, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(rec)$
R compliance (rc)	$I(rc, \boldsymbol{\phi} \mid \boldsymbol{\phi})$	$p(rc)$
Usability (U)	$I(U, \phi \mid und, lrn, oper, attr, uc)$	$p(U \mid und, lrn, oper, attr, uc)$
Understandability (und)	$I(und, \phi \mid \phi)$	$p(und)$
Learnability (lrn)	$I(lrn, \phi \mid \phi)$	$p(lrn)$
Operability (oper)	$I(oper, \boldsymbol{\phi} \mid suit, chg, adpt, inst) \rightarrow$ $I(oper, \phi \mid suit, chg, adpt, inst, E, R)$	$p(oper \mid suit, chg, adpt, inst, E, R)$
Attractiveness (attr)	$I(attr, \phi \mid \phi)$	$p(attr)$
U compliance (uc)	$I(uc, \phi \mid \phi)$	$p(uc)$
Efficiency (E)	$I(E, \phi \mid tb, ru, ec)$	$p(E \mid tb, ru, ec)$
Time behavior (tb)	$I(tb, \phi \mid \phi)$	$p(tb)$
Resource utilization (ru)	$I(ru, \phi \mid \phi)$	$p(ru)$
E compliance (ec)	$I(ec, \phi \mid \phi)$	$p(ec)$
Maintainability (M)	$I(M, \phi \mid anal, chg, stab, test, mc)$	$p(M \mid anal, chg, stab, test, mc)$
Analyzability (anal)	$I(anal, \phi \mid \phi)$	$p(anal)$
Changeability (chg)	$I(chg, \phi \mid \phi)$	$p(chg)$
Stability (stab)	$I(stab, \phi \mid \phi)$	$p(stab)$
Testability (test)	$I(test, \phi \mid \phi)$	$p(test)$
M compliance (mc)	$I(mc, \phi \mid \phi)$	$p(mc)$
Portability (P)	$I(P, \phi \mid adpt, inst, coexist, repl, pc)$	$p(P \mid adpt, inst, coexist, repl, pc)$
Adaptability (adpt)	$I(adpt, \phi \mid \phi)$	$p(adpt)$
Installability (inst)	$I(inst, \phi \mid \phi)$	$p(inst)$
Co-existence (coexist)	$I(coexist, \phi \mid \phi)$	$p(coexist)$
Replaceability (repl)	$I(repl, \phi \mid \phi)$	$p(repl)$
P compliance (pc)	$I(pc, \phi \mid \phi)$	$p(pc)$

to one another and it states that the six attributes have minimal overlap. For the purpose of this research, minimal overlap is taken to have the classical meaning, that is, the attributes are not dependent on one another. Attributes, however, may be conditionally dependent given some other sub-attribute and still have minimal overlap. Using the first interpretation, it is unclear which aspects (or sub-attributes) of *Functionality, Reliability*, and *Efficiency* will affect *Usability* without guidance or further information. The only other causal phrase that provides minimal insight

Table 2 ISO/IEC 9126-1 causal list and CPDs for quality-in-use attributes.

Node (symbol)	CIS	CPD
Effectiveness (Eff)	$I(Eff, \phi \mid \phi)$	$p(Eff)$
Productivity (Prod)	$I(Prod, \phi \mid \phi)$	$p(Prod)$
Safety (Safety)	$I(Safety, \phi \mid F, R, U, M)$	$p(Safety \mid F, R, U, M)$
Satisfaction (Satisf)	$I(Satisf, \phi \mid \phi)$	$p(Satisf)$

is clause 6.3.3 Note 1 (aspects of *suitability, changeability, adaptability*, and *installability* may affect *operability*). In clause 6.3.3 Note 1, *suitability* (a sub-attribute of *Functionality* as stated in clause 6.1) may affect *operability* (a sub-attribute of *Usability* as stated in clause 6.3). This relationship appears to substantiate the first interpretation since an aspect (*suitability*) of *Functionality* affects *operability* and *operability* influences/affects *Usability*. This compound relationship can be shortened and interpreted to mean "an aspect of *Functionality* affects *Usability*." Using the "aspect of *Functionality"* pattern for the other attributes in clause 6.3 Note 1 yields both *Reliability* and *Efficiency* affecting *operability*. The advantage of this interpretation is that it does not violate the major overarching causal phrase in clause 6. The six attributes have minimal overlap through conditional dependence given their sub-attributes and not through direct attribute dependence.

Interpretation #2 (take literal meaning of causal phrases despite ambiguity by weighing all phrases equally)

The second interpretation takes the meaning of causal phrases literally and does not attempt to reconcile competing and conflicting statements. Using this interpretation, the attributes in clause 6.3 Note 1 (*Functionality, Reliability*, and *Efficiency*) affect the attribute *Usability* directly despite the wording that "some aspects of *Functionality, Reliability*, and *Efficiency* will also affect *Usability*." This represents direct causal dependency structures from the attributes of *Functionality, Reliability* and *Efficiency* to the attribute *Usability*. These causal dependency structures are in conflict with clause 6 (external/internal quality is characterized by 6 attributes—*Functionality, Reliability, Usability, Efficiency, Maintainability*, and *Portability*—that have minimal overlap). Because of this conflict, interpretation #2 was not selected.

Further clarification would be required from the authors of the quality standard to determine which interpretation was intended. However, for the purposes of this research, the first interpretation was analyzed, selected and applied. Using interpretation #1, the reconciliation of these relations led to the modified CIS for *operability* shown in Table 1. Despite the slight differences in meaning, both interpretations lead to meaningful graphical representations that are directed acyclic graphs. The reconciliations of both *suitability* and *operability* are indicated by the symbol "→" and are shown in the second column of Table 1.

E. Step 4

Using the causal list, structural forms of CPDs should be generated and aggregated to form the JPD structure. Each CIS implies a conditional probability distribution of the form

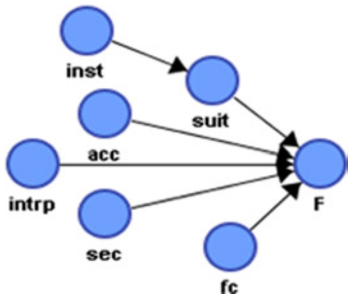
$$I(X_i, \boldsymbol{\phi}|\pi_i) \Longleftrightarrow p(X_i|\pi_i) \quad i = 1, \ldots, n \tag{5}$$

The CIS for each node of the ISO/IEC 9126-1 software standard is represented as a CPD in the third columns of Tables 1 and 2. Aggregation of all of the CPDs for the nodes forms the causal structure of the following joint probability distribution using the factorization in equation (3):

$$\begin{aligned} JPD = {} & p(F|suit, acc, intrp, sec, fp)p(suit|inst)p(acc)p(intrp)p(sec)p(fc) \\ & \times p(R|mat, ft, rec, rc)p(mat)p(ft)p(rec)p(rc) \\ & \times p(U|und, lrn, oper, attr, uc)p(und)p(lrn)p(oper|suit, chg, adpt, inst, E.R)p(attr)p(uc) \\ & \times p(E|tb, ru, ec)p(tb)p(ru)p(ec) \\ & \times p(M|anal, chg, stab, test, mc)p(anal)p(chg)p(stab)p(test)p(mc) \\ & \times p(P|adpt, inst, coexist, repl, pc)p(adpt)p(inst)p(coexist)p(repl)p(pc) \\ & \times p(\mathit{Eff})p(\mathit{Prod})p(\mathit{Safety}|F, R, U, M)p(\mathit{Satisf}) \end{aligned} \tag{6}$$

F. Step 5

Now, the JPD should be represented graphically allowing only one node for each attribute/sub-attribute to form the dependency model. This step can be simplified by concatenating the graphical representations of the CPDs for each attribute and its associated sub-attributes according to their causal links. Graphical CPD representations for *Functionality* and *Usability* are shown in Fig. 5. Using this process, the complete attribute dependency model for ISO/IEC 9126-1[1] is shown in Fig. 6.



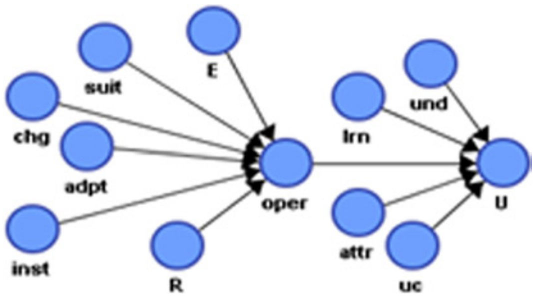


Fig. 5 Graphical CPD representations for functionality and usability.

G. Step 6

The dependency model should be checked for cycles; if cycles exist, the model is not a DAG (thus, discard); if cycles do not exist, then the model is a DAG. A DAG is a directed graph that contains no cycles. The check for cycles tells us whether the dependency model will be appropriate as a Bayesian network since these networks are probabilistic models restricted to DAGs. One way to test for cycles in the dependency model is by way of a multi-level representation.[32] The basic idea of a multi-level representation is to organize the nodes on different levels in such a way that there is no link between nodes on the same level. In this representation, every node is connected to some other node on different level. In the case of DAGs, a multi-level representation is one in which all the arrows in the dependency model point in the same direction. A directed multi-level representation can be used to check for cycles in a dependency graph since a multi-level representation is only possible if the graph is a DAG.[32]

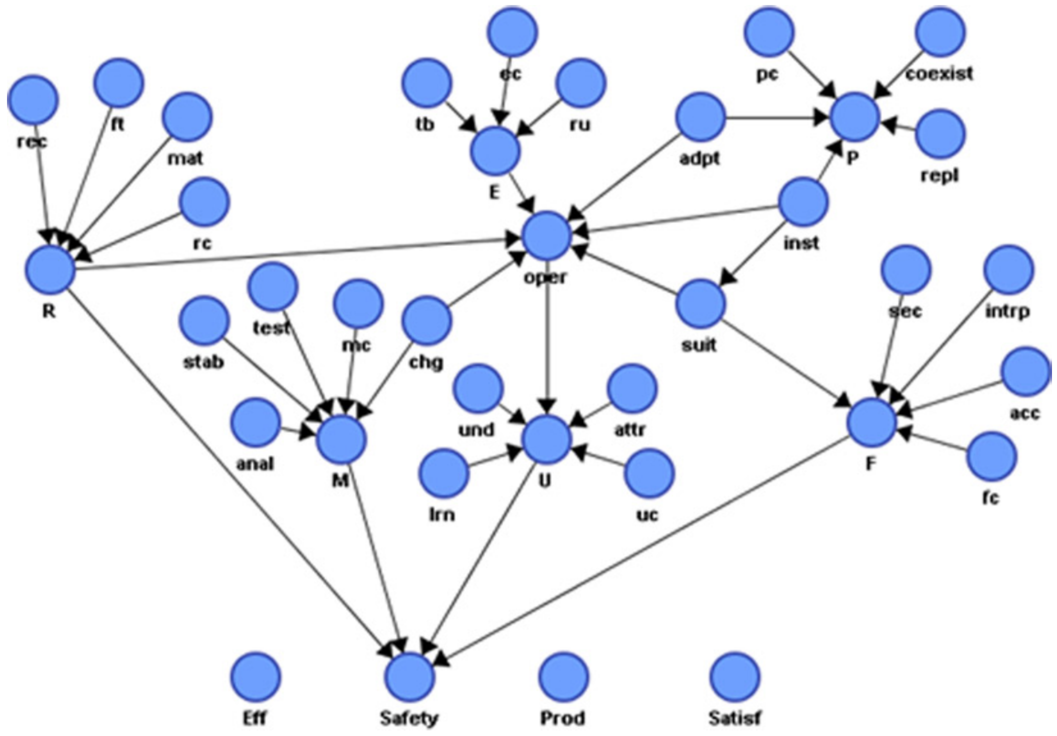


Fig. 6 ISO/IEC 9126-1 dependency model (graphical JPD representation).

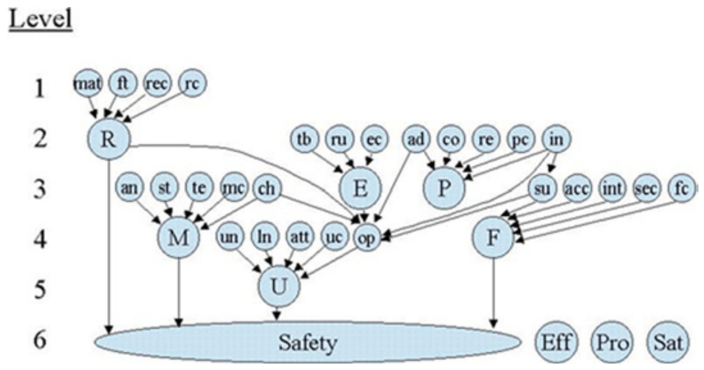


Fig. 7 Multi-level representation of ISO/IEC 9126-1 dependency model.

Figure 7 reveals the directed multi-level representation of the ISO/IEC 9126-1 dependency model in Fig. 6. Since it is possible to depict this representation, the attribute dependency model contains no cycles and is considered a DAG.

H. Output

The output of the extraction process has revealed a directed network topology that will serve as the dependency structure of a Bayesian network. This dependency structure of the BN represents the relationships between the attributes/sub-attributes of the ISO/IEC 9126-1[1] software standard. The ISO/IEC 9126-1 DAG is shown in Fig. 6. Depending on the attributes of interest to the user, the dependency model (Fig. 6) or its equivalent JPD in equation (6) can be marginalized leaving only the required or desired attributes/sub-attributes.

One distinct advantage of the dependency model representation of the ISO/IEC 9126-1 software standard is the ability to graphically view the global set of relationships. This graphical representation describes the complexity of the attribute relationships. It can also be used by standards-making bodies to verify understandings between attributes, to clarify the reasoning employed for such independence relationships and to resolve misunderstandings between member states as a means to achieving consensus. The extraction process can also be used to graphically reveal any attribute relationship incoherency expressed within the software standard.

V. Conclusion

A process has been presented that extracts causal relationships from software product quality standards and equivalently depicts them as an acyclic dependency model if it is possible. This dependency model can then be used for probabilistic evaluation of COTS software products using historical COTS software data. The extraction of dependency models from software standards provides a means of establishing a state-of-the-art criterion for software metric comparison since the dependency model incorporates well-defined attribute dependency relationships and provides formalisms for handling uncertainty. From a larger perspective, the dependency model can be viewed as a graphical representation of the knowledge contained in the software standard. This, in essence, is a knowledge base of software attributes and their relationships developed by expert consensus. Standards making bodies may also use this process to clarify the reasoning, justifications and causal implications they employ when developing standards.

References

[1]*ISO/IEC, Software Engineering—Product Quality—Part 1: Quality Model*, ISO/IEC 9126-1, June 2001.

[2]Morris, A. T., "A Bayesian Network-Based Scoring Methodology for COTS Software," Ph.D. Dissertation, Department of Systems and Information Engineering, University of Virginia, Charlottesville, VA, May 2004.

[3]Vliet, H. V., *Software Engineering: Principles and Practice,* 2nd ed., Wiley & Sons, Inc., West Sussex, England, 2000.

[4]SEPG 100.0, *Procedure for Software Planning*, NASA Langley Research Center, Hampton, VA, November 1988.

[5]NASA Policy Directive 2820.1, NASA Software Policies, Washington, DC, May 1988.

[6]Abts, C., "COTS Software Life Cycle Cost Modeling," Ph.D. Thesis, University of Southern California, 1999.

[7]Basili, V. R., and Boehm, B., "The COTS-Based Systems Top 10 List," *Software Management*, May 2001, pp. 91–93.

[8]*Managing the Use of Commercial-off-the-Shelf (COTS) Software Components for Mission Critical Systems*, AIAA Guidebook G-118-2006, AIAA, Reston, VA, 2006.

[9]Popov, P., Riddle, S., et al., *On Systematic Design of Protectors for Employing OTS Items*, Center for Software Reliability, City University and Newcastle University, UK, 2001.

[10]*Shuttle Master Verification Plan*, NSTS 0770010-MVP, Vol. IX, Part 1, Rev. B, NASA JSC, TX.

[11]Friedman, M. A., and Voas, J. M., *Software Assessment: Reliability, Safety, Testability*, John Wiley & Sons, New York, NY, August 1995.

[12]Fenton, N. E., *Software Metrics: A Rigorous Approach*, Chapman and Hall, London, 1991.

[13]Mosley, V., "How to Assess Tools Efficiently and Quantitatively," *IEEE Software*, May 1992, pp. 29–32.

[14]Poston, R., and Sexton, M., "Evaluating and Selecting Testing Tools," *IEEE Software*, May 1992, pp. 33–42.

[15]*ISO/IEC*, *Information Technology—Software Product Evaluation*, ISO/IEC 9126, 1991.

[16]Gilb, T., *Principles of Software Engineering Management*, Addison Wesley, 1987.

[17]Kitchenham, B., "Towards a Constructive Quality Model. Part I: Software Quality Modeling, Measurement and Prediction," *Software Engineering Journal*, July 1987, pp. 105–113.

[18]Briand, L. C., "COTS Evaluation and Selection," *Proceedings of the International Conference on Software Maintenance*, Bethesda, Nov. 1998, IEEE Comput. Soc., Los Alamitos, pp. 222–223.

[19]Cardenas-Garcia, S., and Zelkowitz, M., "A Management Tool for Evaluation of Software Designs," *IEEE Transactions on Software Engineering*, Vol. 17, No. 9, Sept. 1991, pp. 961–971.

[20]Carney, D. J., and Wallnau, K. C., "A Basis for Evaluation of Commercial Software," *Information and Software Technology* Vol. 40, 1998, pp. 851–860.

[21]Dick, R., and Hunter, R., "Subjective Software Evaluation," *Software Quality Management II. Building Quality into Software 2*, 1994, pp. 321–334.

[22]Jeanrenaud, A., and Romanazzi, P., "Software Product Evaluation Metrics: Methodological Approach," *Software Quality Management II. Building Quality in Software 2*, 1995, pp. 59–69.

[23]Kontio, J., *OTSO: A Systematic Process for Reusable Software Component Selection*, CS-TR-3478, University of MD Technical Reports, University of Maryland, College Park, MD, 1995.

[24]Saaty, T., *The Analytical Hierarchy Process*, McGraw-Hill, New York, 1990.

[25]Morisio, M., and Tsoukias, A., "IusWare: A Methodology for the Evaluation and Selection of Software Products," *IEEE Proceedings of Software Engineering*, Vol. 144, No. 3, June 1997, pp. 162–174.

[26]Shin, H., and Lee, J., "A Process Model of Application Software Package Acquisition and Implementation," *Journal of Systems and Software*, Vol. 32, 1996, pp. 57–64.

[27]Tran, V., Liu, D., Hummel, B., "Component-Based Systems Development; Challenges and Lessons Learned," *Proceedings of the Eighth IEEE International Workshop on Software Technology and Engineering Practice Incorporating Computer Aided Software Engineering*, London, July 1997, IEEE Comput. Soc., Los Alamitos, pp. 452–462.

[28]Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[29]Campos, L. M., "Characterizations of Decomposable Dependency Models," *Journal of Artificial Intelligence Research*, Vol. 5, 1996, pp. 289–300.

[30]Shachter, R. "Probabilistic Inference and Influence Diagrams," *Operations Research*, Vol. 36, 1988, pp. 589-604.

[31]Cooper, G. F., "The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks," *Artificial Intelligence*, Vol. 42, 1990, pp. 393-405.

[32]Castillo, E., Gutierrez, J. M., and Hadi, A. S., *Expert Systems and Probabilistic Network Models*, Springer-Verlag, New York, 1997.

James Hargrave
Associate Editor