Revealing the ISO/IEC 9126-1 Clique Tree for COTS Software Evaluation

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

Previous research has shown that acyclic dependency models, if they exist, can be extracted from software quality standards and that these models can be used to assess software safety and product quality. In the case of commercial off-the-shelf (COTS) software, the extracted dependency model can be used in a probabilistic Bayesian network context for COTS software evaluation. Furthermore, while experts typically employ Bayesian networks to encode domain knowledge, secondary structures (clique trees) from Bayesian network graphs can be used to determine the probabilistic distribution of any software variable (attribute) using any clique that contains that variable. Secondary structures, therefore, provide insight into the fundamental nature of graphical networks. This paper will apply secondary structure calculations to reveal the clique tree of the acyclic dependency model extracted from the ISO/IEC 9126-1 software quality standard. Suggestions will be provided to describe how the clique tree may be exploited to aid efficient transformation of an evaluation model.

Nomenclature

CBS	=	COTS-based systems
COTS	=	Commercial off-the-shelf
CPD	=	conditional probability distribution
DAG	=	directed acyclic graph
DOD	=	Department of Defense
$\boldsymbol{G}$	=	a graph of topological dependence structure
$I(.,.\|.)$	=	a conditional independence rule
ISO/IEC	=	International Organization of Standardization/International Electrotechnical Commission
JPD	=	joint probability distribution
$\mathrm{p}(\mathrm{x})$	=	the unconditional probability distribution
$p(x \mid \boldsymbol{\pi}_x)$	=	a conditional probability distribution
$\boldsymbol{\pi}_x$	=	parents of node x
SEI	=	Software Engineering Institute
$\boldsymbol{X}, \boldsymbol{Y}$	=	sets of variables
x, y	=	instantiated variables

I. Introduction

Significant increase in the use of commercial off-the-shelf (COTS) software by DoD and other government and business organizations[1] has led to greater demands being placed on computer-based systems. These demands have basically directed more capability into the software for decision-making including the ability to operate with minimal oversight. Given vendor volatility in the commercial marketplace as well as COTS software obsolescence risks, COTS software products continue to evolve at an ever-increasing rate. This condition requires continual monitoring and evaluation of COTS software products for organizations that seek to mitigate COTS software risks. As the commercial market provides increased diversification of software products (each with uncertain pedigree), there is a critical need for COTS software evaluation techniques to analyze and compare the fit or misfit between competing offers, particularly as an organization chooses to evolve or replace software components. Most COTS

* Software Manager, Safety-Critical Avionics Systems, Mail Stop 130, AIAA Lifetime Associate Fellow.

software evaluation techniques are custom tailored to a client's needs, and hence, are not effective as general acceptance tools. The techniques that are general tend to suffer from three main problems. The first problem involves the inability to accommodate COTS software trade-offs in the requirements acquisition process. Morris[2] has proposed an attribute acceptance paradigm that can effectively bridge the gap between what a client desires and what has been demonstrated via evidence for COTS software evaluations. The second problem involves establishing a set of consistent definitions and qualitative structures for software attributes. Subsequently, there are issues of how to incorporate these attributes to quantify evaluation metrics. Morris and Beling[3] have recently developed a process that extracts attribute dependency models from software quality standards for COTS software evaluation. In their research, the definition and structure of software attributes and their relationships are extracted in the form of a dependency model that can be quantified probabilistically using historical COTS software data in the context of Bayesian causal networks. The third problem involves the inability to effectively evolve an evaluation model over time. For the purpose of conciseness, restarting an evaluation from scratch is considered inefficient. Efficiency, in this context, is the ability to change or modify only a subset of the original evaluation model while retaining the integrity, coherency and consistency of the final solution. The research in this paper augments the previous research and focuses on the investigation of efficient evolution or adaptation of an evaluation model. As in the previous research, the proposed paradigm is intended to be used in a probabilistic Bayesian network context.

The Software Engineering Institute (SEI) has reported that there is no one best evaluation technique for all COTS software evaluations[4]. Within the evaluation timeline, software and system requirements tend to creep and evolve. Every change to the requirements inevitably leads to a new software evaluation. Similarly, a change to an organization's computing architecture will also sometimes require a new software evaluation. The purpose of the research in this paper is to investigate and exploit clique tree structures (revealed through graphical transformation processes) that can provide efficient insight into how to evolve an evaluation model as a means of saving time and rework.

Background information related to COTS software evaluations, the role of software standards, dependency model extraction for software attributes and secondary (clique tree) structures are described in the next section. Section III applies the clique tree transformation process to an international standard to reveal the resulting clique tree. Section IV will investigate possible uses of the clique tree for efficient evolution of an evaluation model.

II. Background

The clique tree transformation process and the investigation into the efficient evolution of evaluation models (to be described in the following sections) were developed within an overall methodology for COTS software scoring[5]. Only the areas of primary interest to clique tree exploitation will be discussed. These areas include the state of COTS software evaluations, the role of software standards, the software attribute dependency model extraction process, and the insight gained from secondary clique structures.

A. COTS Software Evaluations

The demand and availability of COTS software products has increased drastically over the last ten years. These products serve as either stand alone items or as components in larger COTS-based systems (CBS). In the COTS software market, 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[6]. 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[7,8]. 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[9]. In 2001, it has been estimated that 99 percent of all executing computer instructions come from COTS products[10]. 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.

For most types of mission-critical systems, the COTS acceptance process involves establishing systems and software requirements, testing a pool of candidate software products and evaluating the candidates with respect to the requirements. Evaluation results typically provide a ranking of the candidate products and some threshold that identifies the degree to which the products satisfy the requirements. In the COTS software environment, vendors

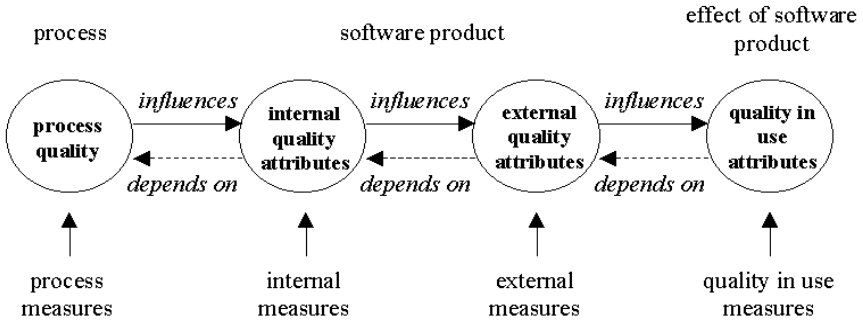

Figure 1. ISO/IEC 9126-1 Software Quality Framework[12].

choose different strategic directions for their existing products. This implies that users of COTS software products must stay informed about changes to the product or at least mitigate the risks to the project if a vendor diminishes product support. In a common scenario, vendors tend to upgrade existing products by supplying patches that may fundamentally change the behavior of the product. For these reasons, users of COTS software products must strategically choose to adopt an evolutionary or adaptive approach for product sustainability, particularly for projects with considerably lengthy life cycles. This implies that software evaluations should occur at different stages of a project life cycle in order to adjust to any number of changes.

The proliferation and volatility of COTS software products reduces software evaluation reliability substantially 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, among others.

There are two areas that hinder efficient adaptation of COTS software evaluation techniques. The first area involves determining metrics, that is, the definition and treatment of software attributes that are consistent, general and flexible for various software evaluations. A proposed solution for this area has been described by Morris and Beling[3]. The second area involves identifying primary or secondary structures and revealing consistent properties to employ when changes are introduced into an evaluation model. The research in this paper proposes an initial step towards this area. Before revealing a possible solution, there must be an explanation describing the role that software standards play in providing consistent terminology and attribute structural relationships for 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. In essence, there are 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[11]. The revision of this international standard, ISO/IEC 9126-1[12], 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 Figure 1). Clear definitions of attributes and supporting sub-attributes are also

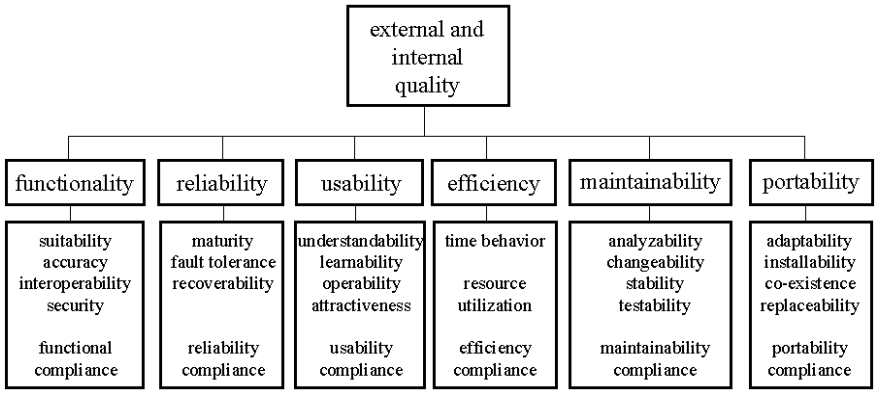

Figure 2. ISO/IEC 9126-1 External and Internal Quality Attributes[12].

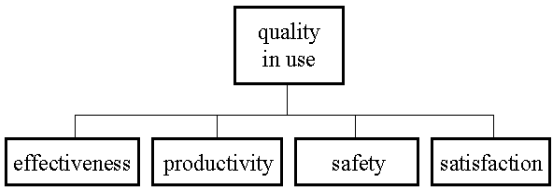

Figure 3. ISO/IEC 9126-1 Quality-In-Use Attributes[12].

provided (see Figure 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 Figure 3).

C. Dependency Model Extraction from Software Quality Standards

As stated earlier, general COTS software evaluation techniques suffer from the lack of use of standardized software metrics, that is, the definition and treatment of software attributes that are consistent, general and flexible for various forms of evaluation. Previous research by Morris and Beling[3] has revealed a process that extracts attribute dependency models from software quality standards for COTS software evaluation. A brief overview of the attribute dependency model will be discussed here since the attribute acceptance paradigm has been designed particularly for such descriptions of attribute relationships.

Dependency is a statement about a set of variables. Formally, a *dependency model*[13] 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 $I(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z})$ is that having observed $\boldsymbol{Z}$, no additional information about $\boldsymbol{X}$ could be obtained by also observing $\boldsymbol{Y}$. In a probabilistic model, $I(\boldsymbol{X}, \boldsymbol{Y} \mid \boldsymbol{Z})$ holds if and only if

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

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[14]. 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.

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[13].

Formally, a 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 $\boldsymbol{X}$,

$$\boldsymbol{P} = \left(p(x_1 \mid \boldsymbol{\pi}_1),\ldots, p(x_n \mid \boldsymbol{\pi}_n)\right) \tag{2}$$

is a set of n conditional probability distributions (CPDs), one for each variable, and $\boldsymbol{\pi}_i$ 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

$$p(x) = \prod_{i=1}^{n} p(x_i \mid \boldsymbol{\pi}_i). \tag{3}$$

When the random variables are discrete, 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[15]. 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.

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. The term *dependency model* will be used to refer only to the causal structure of the relationships among a set of variables.

As stated earlier, Morris and Beling[3] have described a way to extract dependency models from software product quality standards. Since the ISO/IEC 9126-1 standard can be viewed as a knowledge base of software attributes, sub-attributes and their relationships developed by expert consensus, Morris and Beling[3] devised an extraction scheme to reveal the causal relationships using formal mathematically-equivalent representations. Applying their extraction process, the authors revealed a qualitative representation of the attribute relationships within the ISO/IEC 9126-1 standard in the form of the following JPD,

$$\begin{aligned} JPD = {} & p(F \mid suit,acc,intrp,sec,fc)\,p(suit \mid inst)\,p(acc)\,p(intrp)\,p(sec)\,p(fc) \\ & p(R \mid mat,ft,rec,rc)\,p(mat)\,p(ft)\,p(rec)\,p(rc) \\ & p(U \mid und,lrn,oper,attr,uc)\,p(und)\,p(lrn)\,p(oper \mid suit,chg,adpt,inst,E.R)\,p(attr)\,p(uc) \\ & p(E \mid tb,ru,ec)\,p(tb)\,p(ru)\,p(ec) \\ & p(M \mid anal,chg,stab,test,mc)\,p(anal)\,p(chg)\,p(stab)\,p(test)\,p(mc) \\ & p(P \mid adpt,inst,coexist,repl,pc)\,p(adpt)\,p(inst)\,p(coexist)\,p(repl)\,p(pc) \\ & p(Eff)\,p(Prod)\,p(Safety \mid F,R,U,M)\,p(Satisf) \end{aligned} \tag{4}$$

as well as an equivalent graphical representation of a DAG (see Figure 4). Explanations of the symbols for each attribute label or node are found in the previous research[3] and can be easily mapped from Figures 2 and 3. As a general reference, *F, R, U, E, M,* and *P* represent *Functionality, Reliability, Usability, Efficiency, Maintainability,* and *Portability*, respectively (see Figure 2). In like manner, *Eff, Safety, Prod,* and *Satisf* represent *Effectiveness, Safety, Productivity*, and *Satisfaction*, respectively (see Figure 3). The remaining nodes in Figure 4 represent sub-attributes associated with the attributes described.

The graphical and mathematical descriptions of the attribute dependency relationships are crucial for COTS software evaluations. First, the attribute dependency model (mentioned above) describes how the software attributes relate to one another with clear definitions in a concise manner, thereby providing consistency across diverse

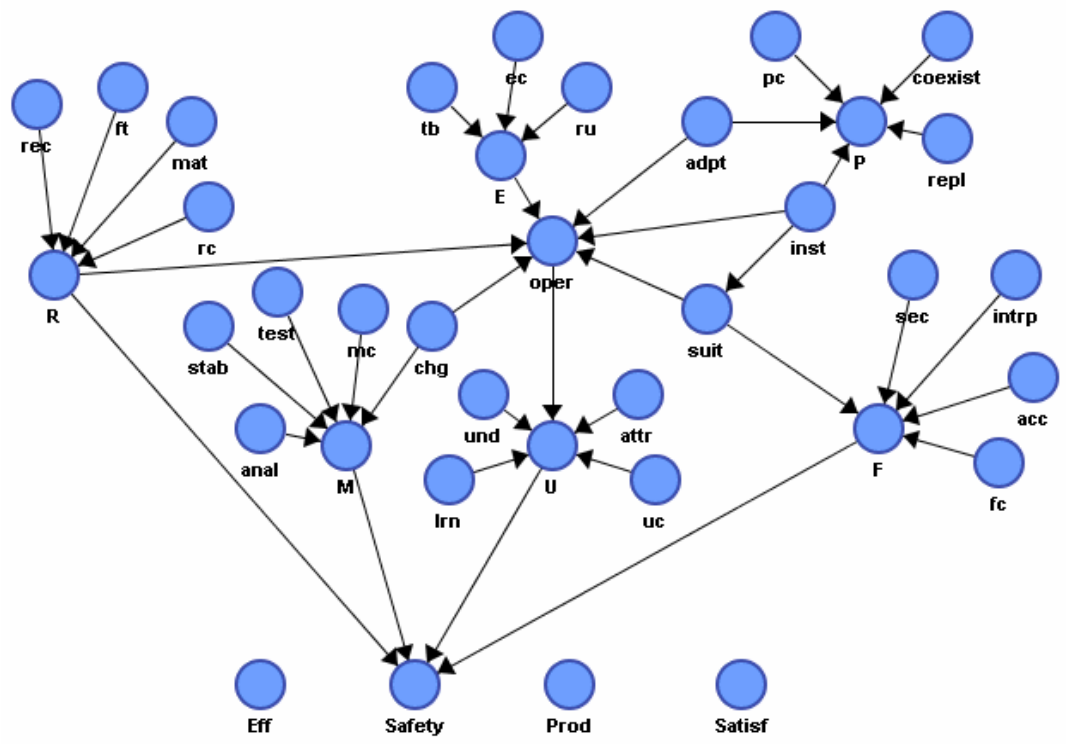

Figure 4. ISO/IEC 9126-1 Dependency Model.

evaluations. Second, the dependency structure between the attribute and sub-attributes provides a coherent aggregation scheme based on the international standard. Third, the incorporation of COTS historical data (similar to data normally collected for evaluation) clustered to the various attributes and sub-attributes can provide a means of producing a quantitative software evaluation based on probability theory. A methodology that uses this approach has been developed by Morris[5] and as such will not be discussed in this paper.

D. Secondary Structures

While experts typically employ Bayesian networks to encode domain knowledge, secondary structures from Bayesian network graphs can be used to determine the probability distribution of any variable *X* using any cluster or clique that contains *X*.

Definition:

Given a Bayesian network over a set of variables $X = \{x_1, \ldots, x_n\}$*, a secondary structure is defined graphically as an undirected tree* τ *where each node in* τ *is a cluster (nonempty set) of variables. The clusters satisfy the join tree property: given two clusters X and Y in* τ*, all clusters on the path between X and Y contain* $X \cap Y$*. For each variable* $x \in X$*, the family of* **X**, $\{X\} \cup \Pi_x$*, where* Π_x *are the parents of node* **X***, is included in at least one of the clusters.*

Secondary structures have been referred to in the literature as *join trees, junction trees, cluster trees*, and *clique trees*. In this research, the term ***clique tree*** will be used to refer to the secondary graphical network.

Clique trees can be built from the DAG of a Bayesian network by applying a series of graphical transformations. These transformations involve a number of intermediate structures. Each step of the clique tree transformation process is briefly described in Figure 5. Steps 4 and 6 (of Figure 5) are nondeterministic; therefore, many different clique trees can be built from the same DAG. Since the establishment of secondary structures is not new[16,17], only a brief explanation of the steps will be provided.

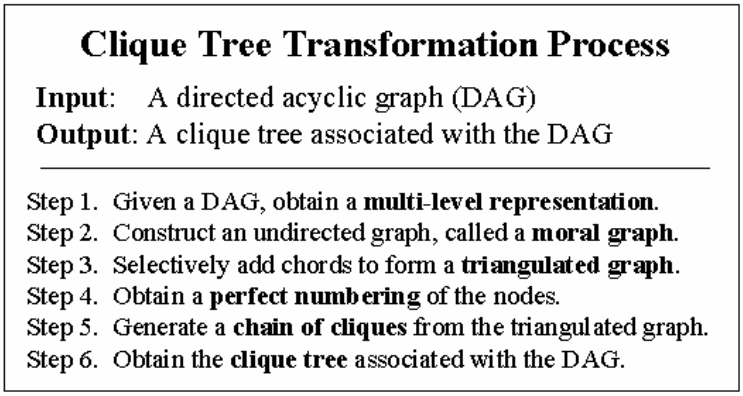

Figure 5. Clique Tree Transformation Process.

A **multi-level representation** is a process of organizing the nodes of a DAG in different levels or layers in such a way that there is no link between nodes on the same level and that every node is connected to some other node in the previous level. A **moral graph** is constructed from the DAG by first joining (adding a link between) every pair of nodes with a common child in the DAG, and then dropping the directionality of the links. Next, an undirected graph is said to be **triangulated**, or chordal, if every cycle of length four or more contains an edge that connects two nonadjacent nodes in the cycle. Various procedures exist for triangulating an undirected graph[16,18]. An optimal triangulation is one that minimizes the sum of the state space sizes of the cliques of the triangulated graph. Optimal triangulation has been shown to be NP-complete[19]. A novel technique to obtain near optimal triangulation is by way of perfect numbering. A given numbering of the nodes of a graph is called a **perfect numbering** if the subsets of the nodes are complete. It has been shown that an undirected graph admits a perfect numbering, if and only if, it is triangulated. The maximum cardinality search algorithm[16] and its variant maximum cardinality search fill-in are techniques used to create triangulated graphs with perfect numbering.

A **clique** in an undirected graph is a sub-graph that is complete (every pair of distinct nodes is connected by an edge) and maximal (the clique is not contained in a larger, complete sub-graph). It has been shown that an undirected graph has an associated **chain of cliques** if and only if it is triangulated[16]. Algorithms exist for identifying cliques[16,20]. The present goal is to build an optimal clique tree by connecting the identified cliques. A set of nodes of a graph is called a **cluster**. A cluster graph is called a clique graph if its clusters are the cliques of the associated graph. Furthermore, a clique graph associated with an undirected graph is called a **join graph** if it contains all the possible links joining two cliques with a common node. Hence, join graphs are unique and the set of clusters with a common node forms a complete set. This property guarantees that clusters with common nodes are always connected. Consequently, join graphs are highly connected. Finally, given a set of *n* cliques, **clique trees** are formed by iteratively inserting edges between pairs of cliques until the cliques are connected by *n*-1 edges. It has been shown that an undirected graph has a clique tree if and only if it is triangulated. The application of the clique tree transformation process (to reveal these secondary structures) will be described in the next section.

III. Application: Revealing the Clique Tree of the ISO/IEC 9126-1 Quality Standard

The purpose of this section is to compute the clique tree of the dependency model extracted from the ISO/IEC 9126-1 software product quality standard (see dependency model in Figure 4). Clique tree computations are performed via secondary structures, a transformation process described in Figure 5. This six step process starts with a directed acyclic graph as input and eventually outputs a clique tree associated with the DAG. As described in the previous section, clique trees are computed by obtaining a multi-level representation of the DAG, constructing a moral graph, triangulating the graph, obtaining a perfect numbering of the nodes, generating a chain of cliques, and then revealing the clique tree associated with the original DAG. This six-step process is as follows:

INPUT: The ISO/IEC 9126-1 dependency model (or DAG) depicted in Figure 4.

STEP 1: Given a DAG, obtain a multi-level representation.

The multi-level representation of the ISO/IEC 9126-1 dependency model (in Figure 4) checks the DAG to ensure that there are no cycles in the graph, that is, each node must be connected only to nodes above or below it (the

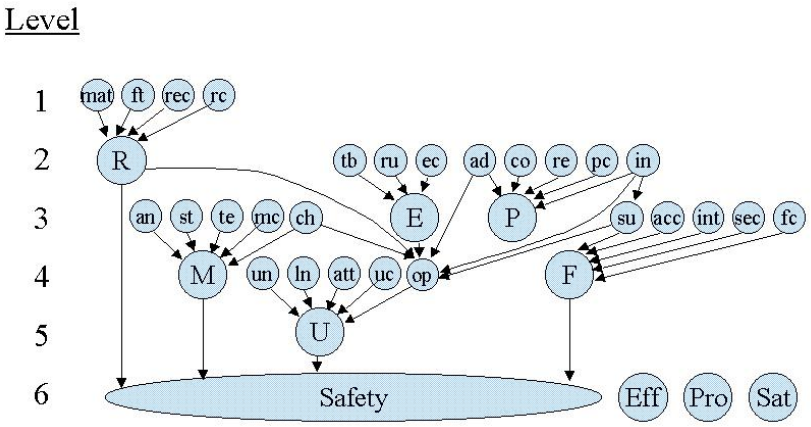

Figure 6. Multi-level Representation of the ISO/IEC 9126-1 Dependency Model.

arrows must face the same direction). Figure 6 reveals the directed multi-level representation of the ISO/IEC 9126-1 dependency model in Figure 4. Since it is possible to depict this representation, the dependency model contains no cycles and is considered a DAG.

STEP 2: Construct an undirected graph, called a moral graph.

The moral graph was constructed by joining every pair of nodes in the DAG with a common child and then dropping the directionality of the links. The moral graph of the ISO/IEC 9126-1 DAG is shown in Figure 7.

STEP 3: Selectively add chords to the moral graph to form a triangulated graph.

The undirected moral graph (see Figure 7) was triangulated by ensuring that every cycle of length four or more contained an edge that connected two adjacent nodes in the cycle. The maximum cardinality search fill-in algorithm[16] was used to triangulate the moral graph. The maximum cardinality search fill-in algorithm is shown in Figure 8. Coincidentally, the moral graph was found to be triangulated, thus Figure 7 also serves as the triangulated graph.

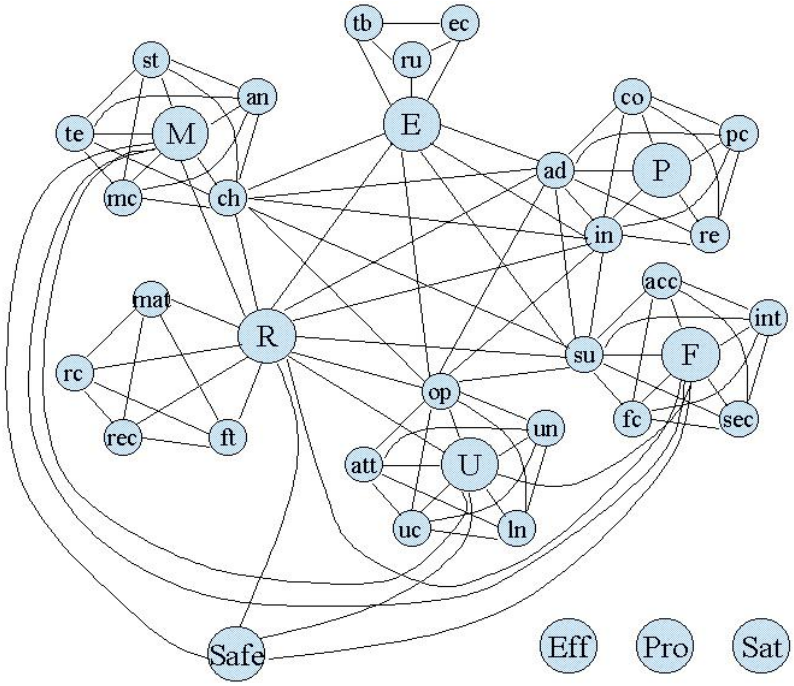

Figure 7. Moral and Triangulated Graph of the ISO/IEC 9126-1 DAG.

Maximum Cardinality Search Fill-In

Input: An undirected graph $G = (X, L)$ and an initial node X_i.

Output: A fill-in L', such that, $G' = (X, L \cup L')$ is a triangulated graph

Initialization Steps:

1. Initially, the fill-in is empty, that is, $L' = \phi$.
2. Let $i = 1$ and assign the first number in the numbering to the initial node X_i, that is, $\alpha(1) = X_i$.

Iteration Steps:

3. An unnumbered node X_k with a maximum number of numbered neighbors is assigned label i, $\alpha(i) = X_k$.
4. If $Nbr(X_k) \cap \{\alpha(1), \ldots, \alpha(i-1)\}$ is not complete, add to L', the necessary links to make this set complete and go to Step 2; otherwise, go to Step 5.
5. If $i = n$, then stop; otherwise, let $i = i + 1$ and go to Step 3.

Figure 8. Maximum Cardinality Search Fill-in Algorithm[16].

STEP 4: Obtain a perfect numbering of the nodes.

A by-product of the maximum cardinality search fill-in algorithm is a perfect numbering of the nodes. The perfect numbering for the nodes in the ISO/IEC 9126-1 DAG is shown in Table 1.

Table 1. A Perfect Numbering of the Nodes in the ISO/IEC 9126-1 DAG.

No.	Node	No.	Node	No.	Node	No.	Node
1	R	11	Op	21	Safe	31	Att
2	Ft	12	Co	22	U	32	Tb
3	Mat	13	Pc	23	F	33	Ru
4	Rec	14	Re	24	Acc	34	Ec
5	Rc	15	P	25	Int	35	Eff
6	Ch	16	M	26	Sec	36	Pro
7	E	17	An	27	Fc	37	Sat
8	Ad	18	St	28	Un		
9	In	19	Te	29	Ln		
10	Su	20	Mc	30	Uc		

STEP 5: Generate a chain of cliques from the triangulated graph.

The cliques of the triangulated graph are shown in Table 2. Using the perfect numbering of the nodes, the chain of cliques was generated by assigning to each clique the largest perfect number of its nodes and then, by arranging the cliques in ascending order (where ties were broken arbitrarily). The algorithm used to generate the chain of cliques is shown in Figure 9.

Generating a Chain of Cliques

Input: A triangulated undirected graph $G = (X, L)$.

Output: A chain of cliques $(C_1,..., C_m)$ associated with G .

1. *Initialization* , choose any node to serve as an initial node, then obtain a perfect numbering of the nodes, $X_1,..., X_n$.
2. Calculate the cliques of the graph, C.
3. Assign to each clique the largest perfect number of its nodes.
4. Order the cliques, $(C_1,..., C_m)$, in ascending order according to their assigned numbers (break ties arbitrarily).

Figure 9. Chain of Cliques Algorithm[16].

Table 2. Chain of Cliques for the ISO/IEC 9126-1 DAG.

Clique set	largest perfect #	ascending order
clique ={ R-mat-rc-rec-ft }	5	C_1
clique ={ M-ch-an-st-te-mc }	20	C_4
clique ={ E-tb-ru-ec }	34	C_8
clique ={ P-ad-in-co-re-pc }	15	C_3
clique ={ F-su-acc-int-sec-fc }	27	C_6
clique ={ U-op-att-uc-un-ln }	31	C_7
clique ={ ch-E-ad-in-su-op-R }	11	C_2
clique ={ Safe-M-R-U-F }	23	C_5
clique ={ Eff }	35	C_9
clique ={ Pro }	36	C_{10}
clique ={ Sat }	37	C_{11}

STEP 6/OUTPUT: Obtain the clique tree associated with the DAG.

A clique tree was obtained by applying the Generate a Join Tree algorithm[16]. This algorithm (see Figure 10) basically lists the cliques in descending order. For each clique, a clique is chosen from the remaining cliques. This chosen clique must have a maximum number of nodes in common with the current clique. The current clique is then linked to the chosen clique. A clique tree results by connecting the cliques with the identified links established according to this process. The clique tree for the ISO/IEC 9126-1 DAG is shown in Figure 11.

Generating a Join Tree

Input: A triangulated undirected graph $G = (X, L)$.

Output: A join (clique) tree $G' = (C, L')$ associated with G .

1. *Initialization* , obtain a chain of cliques of graph G , $(C_1,..., C_m)$.
2. For each clique $C_i \in C$, choose from $\{C_1,..., C_{i-1}\}$ a clique C_k with a maximum number of common nodes and add a link $C_i - C_k$ to L' (initially empty). Break ties arbitrarily.

Figure 10. Generate a Join Tree Algorithm[16].

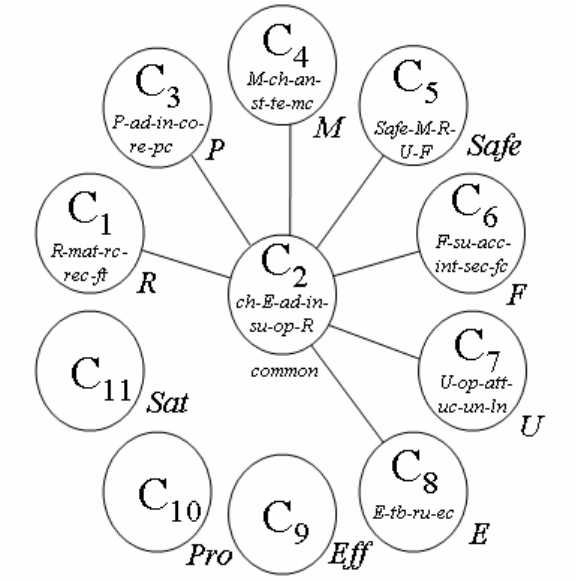

Figure 11. Clique Tree for the ISO/IEC 9126-1 DAG.

IV. The Current and Future Investigation of Clique Fungibility

It is the goal of this research to find ways of exploiting this secondary clique tree structure to reveal an efficient means of modifying an evaluation model due to changes to the requirements (as manifested by changes to required attributes or sub-attributes). Computation of the clique tree can be viewed as a way to prepare the evaluation for future evolution by identifying the underlying structural relationships between the cliques of the dependency model. This section will discuss initial conclusions gained via the investigation of secondary structures. Future work will also be described.

Different or modified software attributes are generally caused by changes to the requirements. When a client changes the software requirements to modify the number of attributes or sub-attributes, the client is redesigning the attribute dependency model. Depending on the degree of changes required, the redesign process could range from deleting a single attribute or sub-attribute to modifying and exchanging complete attribute sets. The redesign process for attribute modification is generally called clique fungibility. ***Clique fungibility*** is the process by which a client can add, delete, or exchange attributes (organized in the form of cliques) of a previous evaluation model to obtain a modified evaluation model. In this research, the term 'evaluation model' is equivalent to a Bayesian network attribute dependency model.

The condition that authorizes clique fungibility is the use of acyclic, conditionally independent attributes and sub-attributes that span the space of decision variables. The variables form the underlying dependency structure of the evaluation model. Clique fungibility is performed by way of clique trees. As discussed earlier, clique trees are obtained using the algorithm shown in Figure 5 and are revealed using the process of secondary structures. Clique trees via secondary structures are highly dependent on the dependency model extracted from the software product quality standard. Redesign via cliques is advantageous in that it requires minimal redesign effort and is computationally efficient for simple to moderate clique exchanges.

In general, there are two types of cliques in clique trees, specific cliques associated with particular attribute clusters and cliques held in common (caused by the clustering of influences between attributes). Clique fungibility involves modifying only specific cliques associated with attributes not common cliques. For instance, the clique tree extracted from the ISO/IEC 9126-1 dependency model (Figure 4) is shown in Figure 11 and annotated in Figure 12. In Figure 11, each clique is identified with a major attribute within the clique. All cliques in Figure 11 have a major attribute identified except for C_2, which is the common clique. In Figure 12, clique C_1 is associated with the attribute Reliability and all of its sub-attributes (see Figures 2 and 4). Clique C_2 is the only common clique in the tree. Attribute clusters are cliques that contain the attribute and all of its associated sub-attributes and nothing more. Although the attribute "R" is found in three distinct cliques (C_1, C_2, and C_5) in Figure 11, only clique C_1 is an attribute cluster. The mapping of dependency model attributes and sub-attributes to cliques is provided in Table 2.

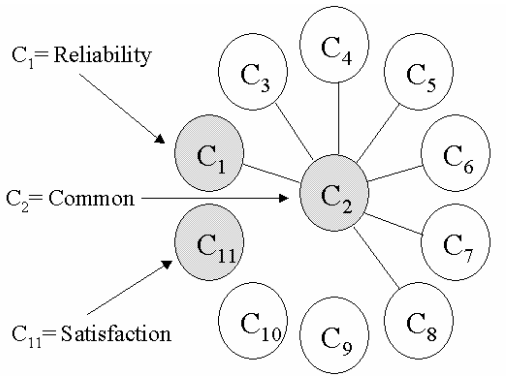

Figure 12. Annotated ISO/IEC 9126-1 Clique Tree.

The following discussion will describe the current investigation on how to discard and add sub-attributes within a clique and how to discard and add cliques associated with attribute clusters.

Discarding sub-attributes within a specific clique associated with an attribute

Discarding a sub-attribute can be performed in one of two ways. First, if the sub-attribute is confined to only one clique, then it can be marginalized from the JPD of the dependency model (see equation 4). The client could restart the evaluation and then determine the new dependency model. The moral and triangulated graphs of the JPD would change slightly, but the same clique tree would result. Using secondary structures reveals that sub-attributes that are confined to only one clique have very limited effect on the remaining cliques and therefore, can be discarded (or marginalized from the dependency model) without significant affect to the evaluation model. For instance, the sub-attribute "fc" or "functional compliance" within clique C_6 (sees Figure 11) can be easily discarded since it only exists within one clique, C_6.

On the other hand, if the sub-attribute resides in more than one clique (like "su" for "suitability" in clique C_6), then it resides in at least one common clique. The resolution of this condition is an area of future research. Initial findings indicate that the simply marginalization of the sub-attribute out of the JPD causes the clique tree to change. The clique tree transformation process (Figure 5) would be performed again to reveal the modified clique tree of the marginalized JPD. It is the goal of future research to find an efficient way of discarding sub-attributes resident within more than one clique.

Adding sub-attributes to a specific clique associated with an attribute

Adding a sub-attribute to a specific clique associated with an attribute follows very similarly to the previous explanation. Instead of discarding a sub-attribute, this process would add a sub-attribute not currently present in the dependency model. If the additional sub-attribute were linked to more than one attribute cluster, then the clique tree would change. Otherwise, the clique tree will remain the same. The resolution of this procedure is also an area of future research.

Discarding cliques associated with attribute clusters

Cliques associated with attribute clusters can be discarded by marginalizing out the attribute and all of its associated sub-attributes from the JPD of the evaluation model. If the clique was linked to any common clique, then all nodes shared in common would have to be marginalized from the common clique. Though this is a technical solution, it is not viewed as efficient. Future research will be used to reveal an efficient procedure, particularly when the common clique shares sub-attributes with the discarded attribute cluster.

Adding cliques associated with attribute clusters

Using a slightly modified process, adding cliques is performed by adding the desired attribute and its associated sub-attributes into the JPD. The resolution of this procedure is also an area of future research.

Clique fungibility (exchanging cliques associated with attribute clusters)

By definition, the process of discarding one clique and replacing it with a different clique is one form of clique exchange or clique fungibility. Future research will investigate the conditions leading to clique fungibility that will allow the efficient adaptation of an evaluation model. It has already been analyzed that independent cliques (like C_9,

C_{10}, and C_{11} in Figure 12) share a common null clique in common; therefore, they can be exchanged with one another without producing change to any other attributes in the JPD. Findings on more complex clique structures are inconclusive thus far. More research is needed to find efficient decomposability properties of secondary structures.

V. Conclusion

Organizations are incorporating more COTS software products into their computer-based systems. Vendors typically make software product decisions based on strategic economic conditions and not necessarily on technical concerns. This implies that users of COTS software products must stay in synch with changes to the product or at least mitigate the risks if a vendor diminishes product support. In both large and small projects alike, software and system requirements tend to creep and evolve over time. Every change to the requirements inevitably leads to a new software evaluation. One way of mitigating these risks is by way of periodic or continual software evaluations. Periodic evaluations due to changes in the requirements or computing system require restarting an evaluation from scratch. The research in this paper investigates and exploits clique tree structures (revealed through graphical transformation processes) that can provide efficient insight into how to evolve an evaluation model as a means of saving time and rework. Efficiency, in this context, is the ability to change or modify only a subset of the original evaluation model while retaining the integrity, coherency and consistency of the remaining portion. The paper proposes the use of secondary structures to describe the underlying relationships between attributes and sub-attributes of an evaluation model. The clique tree transformation process was applied to the ISO/IEC 9126-1 international standard to reveal the resulting clique tree. Initial findings into how to discard cliques from or add cliques to the secondary clique tree structure were provided as well as directions for future research. Throughout the research, the proposed paradigm is intended to be used in a probabilistic Bayesian network context.

References

[1]Ciufo, C. A., "The COTS software market doesn't run on DoD time," *Military Embedded Systems,* Open Systesm Publishing, Fall 2006.

[2]Morris, A. T., "A Probabilistic Software System Attribute Acceptance Paradigm for COTS Software Evaluation," *AIAA's Infotech@Aerospace Conference,* Arlington, Virginia, Sep. 29, 2005.

[3]Morris, A. T., and Beling, P. A., "Extracting Acyclic Dependency Models from Quality Standards for COTS Software Evaluation ," *Journal of Aerospace Computing, Information and Communication,* Vol. 3, July 2006.

[4]Wallnau, Kurt, David Carney, Edwin Morris, Patricia Oberndorf and Charles Buhman, "*A Tutorial on the Theory and Practice of COTS Software Evaluation*," half day tutorial, Symposium on Software Engineering, Pittsburgh, PA, 1998.

[5]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.

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

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

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

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

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

[11]*ISO/IEC, Information Technology – Software Product Evaluation*, ISO/IEC 9126, 1991.

[12]*ISO/IEC, Software engineering - Product quality - Part 1: Quality model*, ISO/IEC 9126-1, June 2001.

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

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

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

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

[17]Huang, C. and Darwiche, A., "Inference in Belief Networks: A Procedural Guide," *Intl. J. Approximate Reasoning*, 15(3), 225-263, 1996.

[18]Kjaerulff, U., "Triangulation of Graphs-Algorithms Giving Small Total State Space," Technical Report R-90-09, Dept. of Math. and Comp. Sci., Aalborg University, Denmark, 1990.

[19]Arnborg, S., Corneil, D. G., and Proskurowski, A., "Complexity of finding embeddings in a k-tree," *SIAM Journal of Algebraic and Discrete Methods*, 8(2), 277, 1987.

[20]Golumbic, M. C., *Triangulated graphs, in Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.