A Probabilistic Software System Attribute Acceptance Paradigm for COTS Software Evaluation

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

Standard software requirement formats are written from top-down perspectives only, that is, from an ideal notion of a client's needs. Despite the exactness of the standard format, software and system errors in designed systems have abounded. Bad and inadequate requirements have resulted in cost overruns, schedule slips and lost profitability. Commercial off-the-shelf (COTS) software components are even more troublesome than designed systems because they are often provided as is and subsequently delivered with unsubstantiated validation of described capabilities. For COTS software, there needs to be a way to express the client's software needs in a consistent and formal manner using software system attributes derived from software quality standards. Additionally, the format needs to be amenable to software evaluation processes that integrate observable evidence garnered from historical data. This paper presents a paradigm that effectively bridges the gap between what a client desires (top-down) and what has been demonstrated (bottom-up) for COTS software evaluation. The paradigm addresses the specification of needs before the software evaluation is performed and can be used to increase the shared understanding between clients and software evaluators about what is required and what is technically possible.

Nomenclature

ABC	=	Acceptable Behavior Constraints
CPD	=	conditional probability distribution
DAG	=	directed acyclic graph
CBS	=	COTS-based system
COTS	=	Commercial off-the-shelf
$\boldsymbol{G}$	=	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*
ROS	=	Rest of the System
SEI	=	Software Engineering Institute (SEI)
$\boldsymbol{X}, \boldsymbol{Y}$	=	sets of variables
x, y	=	instantiated variables

I. Introduction

THE global demand for commercial off-the-shelf (COTS) software has increased to the point where they have become standard components of many of today's mission- and safety-critical systems. As the commercial market provides an increased diversification of software products (each with uncertain pedigree), there is an increased need for COTS software evaluation techniques to analyze and compare the fit or misfit between competing offers. It is important to choose the right COTS product, particularly for safety-critical systems. Most

* Senior Research Engineer, Safety-Critical Avionics Systems, Mail Stop 130, AIAA Lifetime Associate Fellow.

COTS 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 two main problems. The first problem involves the inability to accommodate COTS software trade-offs in the requirements acquisition process. This leads to numerous iterations between the client and the software evaluator to discover and rediscover the client's purpose and intent for the software. Most of the time is spent clarifying the specifications because clients usually do not understand what they need. Another large portion of time is spent in discovery and augmentation mode, that is, the client discovers that some new COTS software capability exists for which he/she had no prior knowledge and commences to augment the requirements to include this new capability. These intensive sessions add considerable time to the requirements acquisition process. 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[1] 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 research in this paper augments the previous research by focusing on the first problem, that is, the development of a probabilistic attribute acceptance paradigm as a means of expressing a client's COTS software needs including the trade-offs that generally accompany the requirements acquisition process. 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[2]. 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. The purpose of the research in this paper is to help reduce the time involved in the iterative requirements process by establishing an acceptance paradigm that incorporates a client's needs and trade-offs before evaluation commences. This new acceptance paradigm requires the client to spend more time in the beginning of the process evaluating their needs and their wants including probabilistic representations of their preference orders and the acceptance or rejection of particular trade-offs. Background information related to COTS software evaluations, the software requirements acquisition process and dependency model extraction for software attributes are described in the next section. Section III provides a description and explanation of the attribute acceptance paradigm and reveals how the paradigm is used in the context of a probabilistic software evaluation process.

II. Background

The attribute acceptance paradigm (to be described in the next section) was one of a number of contributions developed within an overall methodology for COTS software scoring[3]. Only the areas of primary interest to the acceptance paradigm will be discussed. These areas include the state of COTS software evaluations, the problem with software requirements, and the software attribute dependency model extraction process.

A. COTS Software Evaluations

Over the past two decades, there has been a dramatic increase in the demand and availability of COTS software products. 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[4]. 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[5,6]. 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[7]. In 2001, it has been estimated that 99 percent of all executing computer instructions come from COTS products[8]. 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. 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 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.

Software not developed specifically for a project may contain undesirable defects or faults. Some of these faults are caused by improper execution of code; however, the area that produces more faults revolves around the behavior of the software relative to the intent of the requirements. Checking the software's behavior against specified software requirements can generally identify these 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[10]. 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 proliferation 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, etc.

There are two primary issues that hinder the acceptance of the general COTS software evaluation process. The first issue involves the inability to accommodate the COTS software trade-offs in the requirements acquisition process. The second issue 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 the second issue is described by Morris and Beling[1]. The research in this paper proposed a solution for the first issue. Before revealing the proposed solution, there must be an explanation describing why the standard software requirements format is inadequate for COTS software evaluation.

B. The Problem with Requirement Specifications for COTS Software

Requirements acquisition is a process of developing a shared understanding between clients and developers about what is required and what is technically possible[11]. It is, in essence, a trade-off within the design space. Requirements acquisition, in this sense, is an early phase of requirements management and can be partitioned into systems and software components. The goal of software requirements is for a client to determine what they want assuming they know what is currently possible. For the longest time, software requirements have been written deterministically, that is, from a perspective that the required software will perform exactly in a described manner, no more, no less (see Figure 1). Theoretically, this works well for custom-developed software. Despite the exactness of the current format, software and system errors in designed systems have abounded. Bad and inadequate requirements have resulted in cost overruns, schedule slips, frustrated employees, unhappy customers and lost profitability[12]. Raymond Dion of Raytheon found that approximately 40 percent of the total budget for software projects was rework[13]. Barry Boehm, who developed a widely used software cost estimation model, estimates that the cost of rework approaches 50 percent on large software projects[14]. Bell Labs and IBM studies have determined that 80 percent of all product defects are inserted in the requirements definition stage[12]. In general, rework consumes 40 to 50 percent of software project budgets. This means that requirement errors take up to a 42 percent bite out of total software development resources. Hence, any improvement in the requirements definitions phase will produce significant savings in both time and money.

Requirements definition is hard because clients frequently do not understand their own needs or know what they fully want before engaging a developer or software evaluator[15]. To compound the problem, developers make assumptions about client needs and do not check their assumptions before charging into development. Concurrently, customer expectations drift during the development and procurement phases. What are possible solutions to this problem? Sutcliffe has shown that there is an inevitable intertwining between system and software requirements and that the interactions between systems and software should be modeled as well[16].

Some requirements definition researchers view the world as moving and dynamic (uncertain and probabilistic), hence a place where all the variables are not known that leads to the inevitable intertwining between specification and design. Other researchers view the world as closed (deterministic), that is, all the variables that might influence the requirements are assumed known. The current shift to COTS software products embodies the perspective that users do not know or understand all the possible (and perhaps anomalous) behaviors that can occur. This lack of insight leads a user to a more uncertain and hence, probabilistic viewpoint. Various techniques can be used to minimize COTS uncertainties within software requirements. Prevalent techniques involve the use of wrappers and protectors to prevent anomalous COTS software behavior from causing hazardous and unwanted events in complex

Table of Contents

Table of Contents		**ii**
Revision History		**ii**
1.	**Introduction**	**1**
1.1	Purpose	1
1.2	Document Conventions	1
1.3	Intended Audience and Reading Suggestions	1
1.4	Project Scope	1
1.5	References	1
2.	**Overall Description**	**2**
2.1	Product Perspective	2
2.2	Product Features	2
2.3	User Classes and Characteristics	2
2.4	Operating Environment	2
2.5	Design and Implementation Constraints	2
2.6	User Documentation	2
2.7	Assumptions and Dependencies	3
3.	**System Features**	**3**
3.1	System Feature 1	3
3.2	System Feature 2 (and so on)	4
4.	**External Interface Requirements**	**4**
4.1	User Interfaces	4
4.2	Hardware Interfaces	4
4.3	Software Interfaces	4
4.4	Communications Interfaces	4
5.	**Other Nonfunctional Requirements**	**5**
5.1	Performance Requirements	5
5.2	Safety Requirements	5
5.3	Security Requirements	5
5.4	Software Quality Attributes	5
6.	**Other Requirements**	**5**
Appendix A: Glossary		**5**
Appendix B: Analysis Models		**6**
Appendix C: Issues List		**6**

Figure 1. Standard Requirements Format.

computer systems[17]. For this technique, software requirements are considered more deterministic. Behavioral deviations between the COTS software product and the software requirements are compensated by the wrapper/protector. The COTS software/wrapper/protector combination, in this sense, works in concert to fulfill the software requirements. These techniques have their own benefits and liabilities that must be carefully weighed. Another technique recommends employing a highly iterative and interleaved process between requirements elicitation and software evaluation[18]. Some organizations are starting to utilize this highly iterative approach for requirements management.

The research in this paper differs from previous approaches in that it requires the client to specify their needs including the technical performance trade-offs they are willing to make *a priori*, that is, before the software evaluation begins. Current industry practice calls for the developer or evaluator to perform requirements trade-off studies on behalf of the client during and after the acquisition process to help determine the set of requirements that is necessary and affordable. Often such analysis and understanding do not become manifest to the client until after the software has been delivered or evaluated, in which case, of course, it is too late to react effectively[19]. In many cases, the last 10% of requirements can drive a huge increase in cost, a relationship of which the client may be unaware. Indeed, the client may have no appreciation whether this last 10% is even necessary for the intended mission and, thus, not understand the mission may be effectively performed by a software alternative that is much less costly to procure. The research in this paper is less deterministic in that the trade-offs are described in a more probabilistic format. The attribute acceptance paradigm serves to specify desirable functional obligations of the COTS software and system combination with an underlying purpose similar to the *Acceptable Behavior Constraints* (ABC's) espoused by Popov et al[20]. The attribute trade-offs in the software requirements (or attribute acceptance document) will serve the same purpose as the iterative and interleaving process described by Maiden[18]. The primary difference is the distribution of energy dedicated to the process. The highly iterative process[18] requires many discussions with the client and is spread out over time. The requirements format in this research requires very few interactions with the client but is more temporally-demanding initially since it requires the client to describe not only their software requirements but also the technical performance trade-offs they are willing to accept in the decision space *a priori*. This new paradigm is developed to be used primarily in the normative (acceptance testing) context but may be adapted to play a crucial role in formative (design to capabilities of products) evaluations.

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[1] 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*[21] 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[22]. 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[21].

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 discreet, 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[23]. 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.

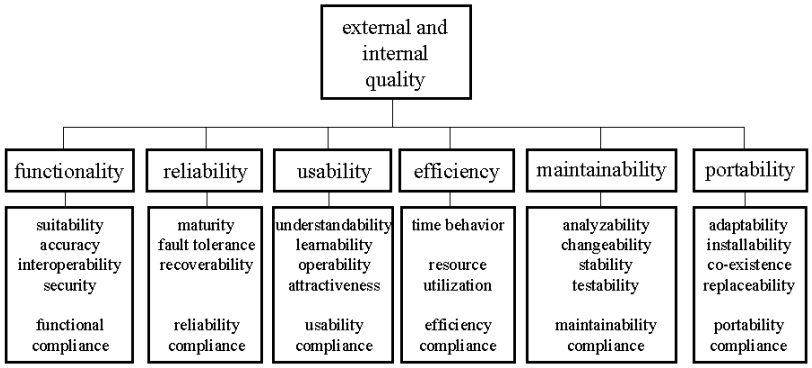


Figure 2. ISO/IEC 9126-1 External and Internal Quality Attributes[24].

In the previous research, Morris and Beling[1] have described a way to extract dependency models from the ISO/IEC 9126-1[24] software product quality standard. 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. 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. Clear definitions of attributes and supporting sub-attributes are also provided (see Figure 2). Information in the standard describes the causal relationships between the attributes and sub-attributes. Using their extraction process, the authors revealed a qualitative representation of the attribute relationships within the ISO/IEC 9126-1 standard in the form of a 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 3). Explanations of the symbols for each attribute label or node are found in the previous research and can be easily mapped from Figure 2. As a general reference, *F*, *R*, *U*, *E*, *M*, and *P* represent *Functionality*, *Reliability*, *Usability*, *Efficiency*, *Maintainability*, and *Portability*, respectively.

The graphical and mathematical descriptions of the attribute dependency relationships are crucial for COTS software evaluations. First, the attribute dependency model describes how the software attributes relate to one another with clear definitions in a concise manner thereby providing consistency across diverse 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[3]. It is important to note that this methodology quantifies attribute metrics from both top-down (form of attribute sub-model is known) and bottom-up formalisms (model form is unknown but will be extracted from available data using data mining techniques). These attribute model formalisms and how they will be used in the attribute acceptance document will be described in the next section.

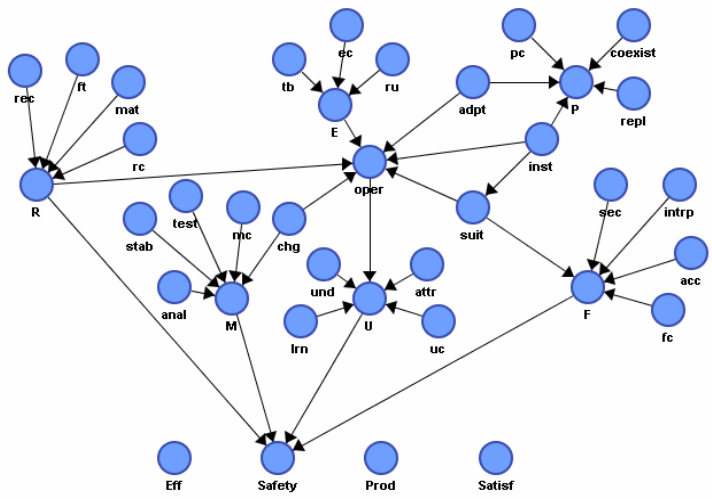


Figure 3. ISO/IEC 9126-1 Dependency Model.

III. The Attribute Acceptance Paradigm

The attribute acceptance paradigm is a format used for a probabilistic COTS software evaluation that allows a client to specify the acceptance or rejection of clearly defined attributes as well as the technical performance trade-offs that are acceptable for the system in which the software will be embedded. This paradigm is designed to be used in a probabilistic software evaluation context (see Figure 4). The following subsections will describe the specific evaluation context with particular emphasis on the attribute acceptance paradigm.

A. Context of Paradigm in a Probabilistic Software Evaluation Process

The diagram in Figure 4 represents the specific probabilistic software evaluation context used in this research. The development of the attribute acceptance paradigm is shown as the shaded set of boxes in Figure 4. As shown in the process, the client selects particular attributes and sub-attributes that are required for the evaluation. The client then determines the type of model that will be developed for each required attribute. Attribute models may be either top-down (the model form is provided by the client) or bottom-up (the model form is extracted from the available data using data mining algorithms). For each attribute, the client then proceeds to express a probabilistic acceptance criterion that describes how much of each sub-attribute will be accepted in combination with the other required sub-attributes. Each probabilistic acceptance criterion is described in the form of a conditional probability distribution. Additionally, the client may include contingencies *a priori* in the acceptance document in the event that some of the COTS software data (used in the overall evaluation) is either missing or latent. The aggregation of the client's needs, trade-offs and contingencies is placed in the attribute acceptance document.

B. Specification of Client Needs, Trade-offs and Contingencies (Attribute Acceptance Document)

The attribute acceptance paradigm is used to express the client's needs, desires, trade-offs, thresholds and contingencies in a probabilistic software evaluation. In order to understand the information contained in the attribute acceptance document, the following explanations will be provided.

1. Software Standard Selection for COTS Software Evaluation

The attribute acceptance paradigm requires a defined set of attributes and sub-attributes. In the previous research, the international standard ISO/IEC 9126-1 was selected. However, any set of software quality characteristics can be used as long as the attributes are formally defined and there is a description of how the attributes relate to one another.

2. The Meaning of a Variable

For software evaluations, there are attribute and sub-attributes. Attributes are quality features of a product decomposed into various sub-attributes whereas sub-attributes are mutually exclusive sub-characteristics of an attribute. For the acceptance paradigm, attributes are treated as binary random variables. For example, let's say that

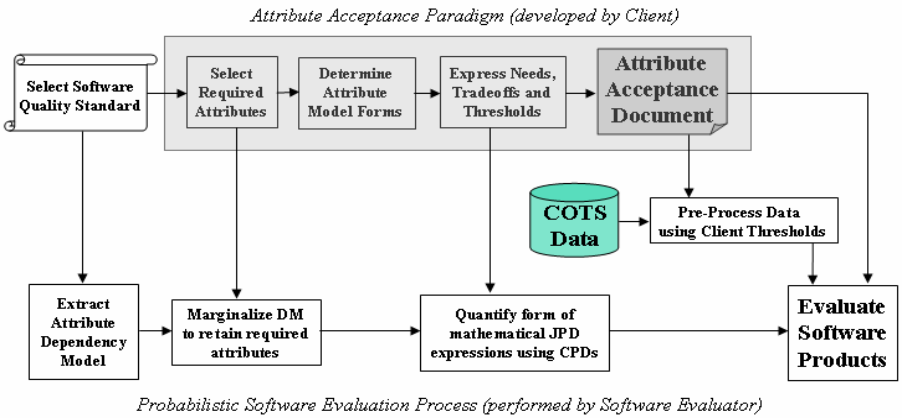


Figure 4. The Probabilistic Software Evaluation Process.

p(Attribute 1 = True) = 1.0. This means that the software product completely meets the client's *attribute 1* expectations. For another certainty condition, let's say that *p(Attribute 1 = True)* = 0.0. This means that the software product does not meet the client's *attribute 1* expectations. On the other hand, for an uncertain condition, like, *p(Attribute 1 = True)* = .8, this means that there is an 80% likelihood that the software product meets the client's *attribute 1* expectations. In this paradigm, observations of sub-attribute random variables are taken from the database of COTS software products. Thus, sub-attributes are associated with observable data and are linked to attributes through conditional probability distributions.

3. The Meaning of Client-Developed Conditional Probability Distributions

When constructed by statistical data algorithms, conditional probability distributions describe how discrete random variables are related to each other in a database given a structure. All the variables are present in the database. CPDs generally take the form of contingency tables with the child node on the right and the parent node(s) on the left. The values of the table elements are the probabilities computed from the database. In Figure 5, the prior distributions of A and B and the structure of how they are related to C are shown. Given this information and the data found in the database, it can be seen that *p(A=blue)* = .5 and *p(B=>5)* = .33. The resulting "standard" CPD that represents the data in the database given the structure of the variables is also shown in Figure 5.

As used in this research, the CPD has a different meaning and is developed by the client and not the data. In the attribute acceptance paradigm, the client-developed CPD describes the type of relationships that are ***acceptable*** between an attribute and its sub-attributes. This relationship is seen as a description of software expectations or an expression of the client's software needs. In other words, the client has some prior knowledge about the system, the interfaces and functionality required as well as the performance needs. The CPD, in this context, explains how the client will adjust his software expectations in light of specific evidence. Higher values map to more favorable expectations. For example, the client knows (from the attribute dependency structure) that the attribute reliability and at least two sub-attributes (maturity and fault tolerance) are required by the COTS software product. However, due to lack of source code visibility, the client can only rely on actual behavior of each COTS software products as evidenced by the data. This data is usually acquired via testing. Given the attribute sub-attribute dependency structure (provided in the software standard), the client can express his reliability needs in the form of a CPD before the evaluation occurs. As seen in Figure 6, the client has provided four statements, namely, if the data from the COTS software product provides evidence revealing that *maturity* is false and *fault tolerance* is false, then this product does not meet my reliability needs or expectations. The second statement in the CPD states that if the COTS data reveals that *maturity* is false and *fault tolerance* is true, then there is an 80% likelihood that the software product will meet my needs. The third statement in the CPD states that if *maturity* is true and *fault tolerance* is false, then there is a 40% likelihood that that the software product meets the required needs. In like manner, the fourth statement in the CPD emphasizes that if software products are *mature* and *fault tolerant* as evidence in the data, then these products completely meet the reliability needs. The combination of statements in this client-developed CPD provides descriptions of levels of acceptance and these statements are probabilistic expressions of the client's software needs. The set of statements can also be used to deduce that fault tolerance is preferred over maturity for the evaluation. The CPD values provided by the client also encapsulate a complete set of preference orders between required sub-attributes. The preference orders can be linear or nonlinear depending on the complexity of the required needs.

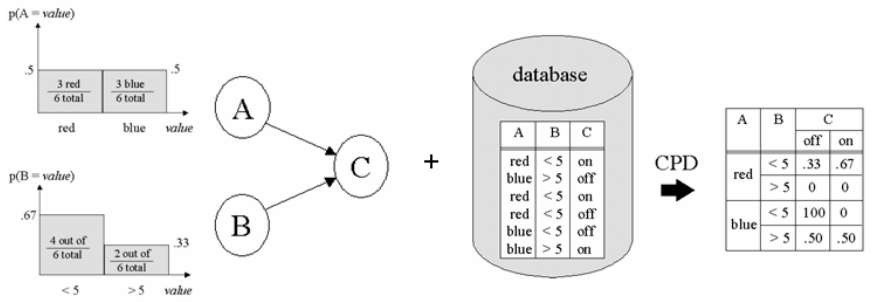


Figure 5. Standard CPD Development.

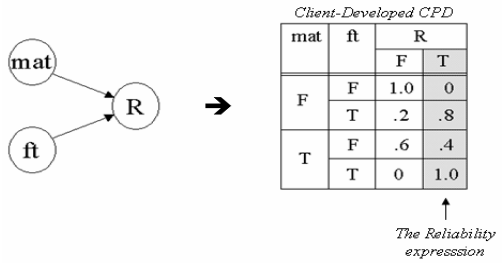


Figure 6. Client-Developed Reliability CPD.

4. Establish Sub-Attribute Thresholds

In order for a software evaluation to be meaningful to a client, threshold values must be established at the beginning of the process. Depending on the criticality of the evaluation, thresholds may be established as reasonable and customary values for required variables or stringent values for high integrity systems. Regardless of the case, these thresholds need to be expressed. In this research, threshold values are expressed by the client for sub-attributes. These threshold values are established before the evaluation begins and are used to tell the software evaluator how to classify the available COTS software data. Thresholds are, in essence, classification mappings from the available range of values in the COTS data to the required range of values used in the evaluation.

For example, the variable *B* in Figure 5 has two possible values, "< 5" and ">5." In this instance, "5" was established as a threshold value for *B*. "Greater than five" is considered one state and "less than five" is considered another state. In actual evaluations, the data available are generally neither clean nor coherent. This means that some type of preprocessing of the data is required to prepare the data to be used in the evaluation. The original data for variable *B*, for example, may have had values ranging from -100 to +100. It is up to the client to determine what threshold value for *B* is required for the evaluation. This is analogous to standard evaluation requirements that express, "the accepted software shall have priority inversion capability." In the attribute acceptance paradigm, it is not known whether priority inversion exists in the data prior to evaluation. But, if it does exist, the values of priority inversion will be set to either "yes" or "no." Other examples of sub-attribute thresholds include:

- Standard requirements ➔ The COTS software must be fault tolerant.
- Paradigm threshold formalism ➔ FT=yes ➔ True and FT=no ➔ False

- Standard requirements ➔ The COTS software shall be stable 90% of the time.
- Paradigm threshold formalism ➔ Stable>=.9 ➔ Good and Stable<.9 ➔ Bad

- Standard requirements ➔ It is desirable for the COTS software to coexist with all legacy software.
- Paradigm threshold formalism ➔ Coexist>=.95 ➔ Yes
  .25<Coexist<.95 ➔ Unsure
  Coexist <=.25 ➔ No

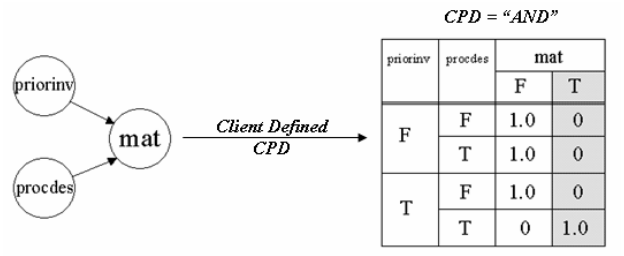


Figure 7. Top-Down Maturity Model.

5. Top-Down Sub-Attribute Models

After the client constructs the CPD for each required attribute as an expression of his software needs, the client then declares what type of sub-attribute model is required for each attribute. There are two types of sub-attribute models, top-down and bottom-up. These models are a way of describing their software needs. Sometimes in an evaluation, needs are very precise and other times they are not. The top-down model (these represent "shall" needs) has the highest priority and requires a specific model form explicitly described by the client. The client also explicitly describes how all the required variables (sub-attributes, sub-sub-attributes, etc.) relate to one another using a set of CPDs. The model is declared before seeing the actual data evidence. Thus, the client is saying that the described relationship is the only relationship that is allowed to meet the software need. Unfortunately, for such stringent requirements, if the evidence is not found in the database, the need cannot be met by the available products and the software evaluation terminates. As seen in the top-down model example in Figure 7, the client (using the *maturity* sub-attribute from the *reliability* attribute in Figure 6) has defined *maturity* specifically as

$$\textbf{\textit{maturity}} = \textit{priority inversion}\ \textbf{AND}\ \textit{process design} \tag{5}$$

The CPD, as defined by the client, serves as an "AND" function. The CPD here states that the *maturity* sub-attribute shall only be defined when *priority inversion* is true AND *process design* is true. The CPD also states that no other evidence combinations for maturity will be allowed by the client. For this top-down model, the client is requiring *priority inversion* and *process design* to be variables in the COTS software data. If these data are not found in the database, then the *maturity* model requirement is not satisfied thus the evaluation terminates because the client requires that this top-down *maturity* model shall be evaluated precisely as prescribed.

6. Bottom-Up Sub-Attribute Models

In bottom-up models, the client allows the sub-attribute models to be extracted from the evidence (data) using some type of statistical data mining algorithm. These models represent software needs that are "nice to have" but not absolutely necessary. Just like top-down models, bottom-up models are described via CPDs before seeing the actual evidence and before performing the software evaluation. This sub-attribute model was added because there are instances where the client may not know the exact model form for sub-attribute acceptance. The client may have a vague sense that the sub-attribute should be included in the evaluation but not the specific form. Figure 8 displays an example of a bottom-up sub-attribute model. In this network, a data set containing evidence on *priority inversion, process design, reentrancy* and *maturity* from a set of COTS software candidates was mined using statistical data mining algorithms to reveal the best model from the available data. For bottom-up models, the data mining algorithm computes the CPD or relationship between the sub-attribute and the remaining model (as performed in Figure 5).

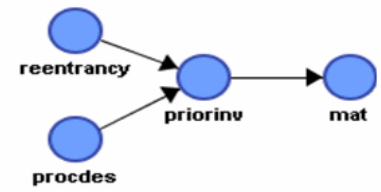
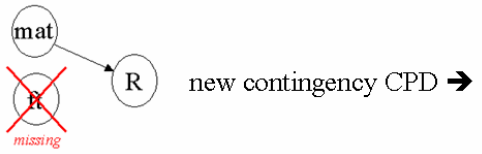


Figure 8. Bottom-Up Maturity Model.



mat	R	
	F	T
F	.6	.4
T	.3	.7

Figure 9. Missing Fault Tolerance Contingency CPD.

mat	fs	R	
		F	T
F	F	1.0	0
	T	.2	.8
T	F	.6	.4
	T	0	1.0

Figure 10. Latent Fail Safe Contingency CPD.

7. Contingency Planning

Because bottom-up models aren't specified exactly, there is flexibility in developing these models from available evidence. This flexibility is described in the form of sub-attribute contingencies. There are two types of *sub-attribute contingencies*, "missing" and "latent." The sub-attribute is "***missing***" means the sub-attribute is not present in the evidence and cannot be substituted by other data. The sub-attribute is "***latent***" means the sub-attribute is not present in the evidence but can be substituted with another data variable or data set. Given the Reliability example described earlier in Figure 6, the client provided an expression of his software needs using a client-developed CPD. This expression was made on the assumption that the evidence "mat" and "ft" would be found in the available evidence. When the client modifies the CPD in the event that "ft" is missing, the client provides a contingency for missing data. In the contingency CPD in Figure 9, the client has decided the following. If "ft" is missing, then average ***True*** and ***False*** values of "ft" to obtain the new contingency CPD. It is up to the client to determine the sub-attribute contingencies before seeing the evidence. The contingency function is selected by the client and, in essence, provides for various trade-offs in the decision space.

In similar fashion, using the same Reliability example from Figure 6, the client can also determine how to modify the original CPD if the sub-attribute fault tolerance or "ft" is latent, that is, missing but can be replaced by another data variable. As seen in Figure 10, the client has decided that if "ft" is latent, then replace "ft" with "failsafe" or "fs" and keep the same CPD values. This contingency allows the software evaluator to replace data labels with other variables that serve similar functions as permitted by the client.

8. The Attribute Acceptance Document

The attribute acceptance document serves as a depository for the client. This document contains the attributes and sub-attributes required for the software evaluation as well as the trade-offs, thresholds, model types and contingencies that are an expression of the client's needs and desires. Figure 11 depicts the generic structure of the attribute acceptance document which differs substantially from the standard requirements format (Figure 1). The first part of the attribute acceptance document (Figure 11) describes the type of software to be evaluated as well as a description of its application and the criticality of its purpose. The second part provides a description of the major partitions of the existing systems and the interfaces that are allowed between the system and the software. Several evaluations establish partitions between the software and the rest of the system (ROS). The third component of the document describes system level characteristics of the evaluation such as the standard that will be used to establish the definition and relationships of the software attributes. From the entire set of attributes, the document determines the set or subset of attributes to be used for the evaluation including the type of model required for each attribute.

The last and largest section of the document details the treatment of each of the required attributes. For each required attribute, the document describes where it is defined, the required sub-attributes and the established thresholds for use with the COTS software data. The types of model that will be used as well as the detailed client-developed CPDs are included. The CPD is provided in the form of a contingency table that describes the client's expectations for the attribute in light of evidence (if it exists). Finally, defined descriptions are provided when top-

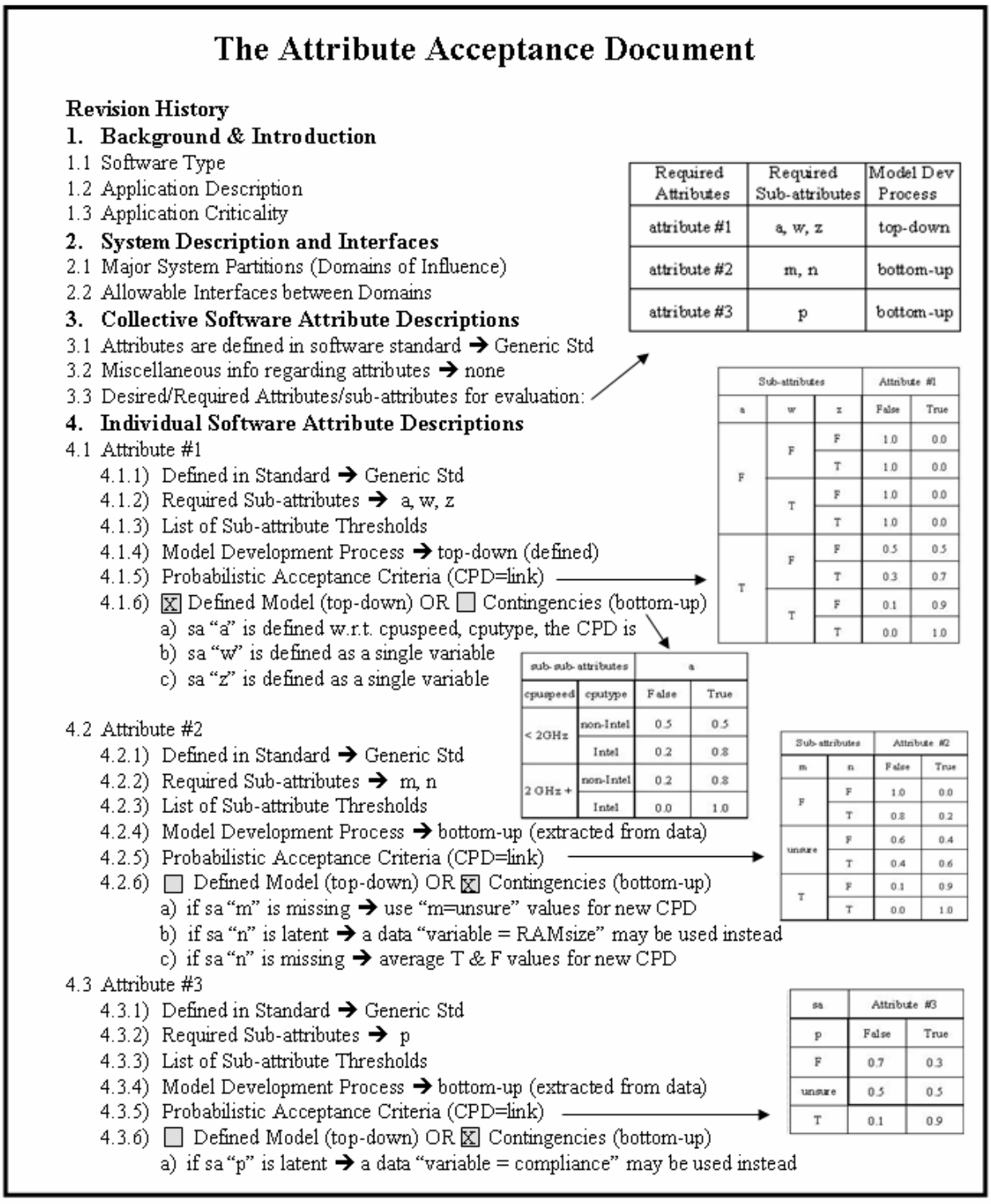


Figure 11. Generic Form of Attribute Acceptance Document.

down models are required and contingencies are provided when bottom-up models are permitted. This document is designed to be used in a probabilistic software evaluation context with the incorporation of COTS software historical data for sub-attributes to provide quantitative comparisons within a pool of COTS software products. It is primarily designed to be used in a normative context but can be modified to play a crucial role in formative evaluations.

C. Collect COTS Software Data

According to the process shown in Figure 4, it is the software evaluator's responsibility to handle and preprocess the historical COTS data. The preprocessing serves to clean and map the data to acceptable ranges required for the COTS software evaluation. The evaluator will use the thresholds requested by the client to do this. These thresholds reside in the attribute acceptance document. It is important to note that evaluation results both rest and rely on the historical data from COTS software products. The term *historical* implies that the data was obtained previously in time through some means and does not represent an estimate or prediction of future behavior. Some software evaluations in the past have been based on theoretical models that predict what a product may or may not due in the future. Because of the intractability of predicting future COTS software behavior, the evaluation process in this research utilizes only actually recorded behavior. A Bayesian method was selected as an aggregation technique to resolve a serious concern in evaluation based on historical data, namely, the ability to update an evaluation once new knowledge has been observed. Bayesian networks have desirable properties of admissibility, that is, as the data set increases, the probability of an evaluation obtaining an optimal product increases. For this reason, the incorporation of COTS historical data is of prime importance. This also includes data collected from software wrapper combination testing as long as the data represent actual observations and not predictive models based on statistics.

Possible sources for COTS software data include individual product testing, vendor-supplied information, legal briefings, customer complaints, product comparisons, software wrapper combination testing as well as information garnered from past users of the product. Optimally, the data should be accurate, complete and mature. Accuracy aims at minimizing incorrect product information. Completeness involves covering several products that span the application domain. Maturity involves providing for a statistically significant product sample over a significant length of time. Optimal conditions are rarely achieved in many real-world evaluations. However, the evaluator should endeavor to acquire as much relevant COTS software product data as possible since the evaluation results and the mined relationships of the sub-attributes rely heavily on the quality and quantity of the data acquired.

D. Perform the COTS Software Evaluation

It is the evaluator's responsibility to perform a detailed software evaluation. The evaluation in this research involves a probabilistic scheme that employs a Bayesian network as a coherent aggregation technique. Details of this approach are provided by Morris[3] and are outside the scope of this paper. Figure 4 describes the major inputs that are required by the evaluator to evaluate the available set of products. By using the attribute acceptance document, the software evaluator should not need months of interactive time with the client. The document, if properly detailed by the client, should provide adequate information for the product evaluation in light of the needs, trade-offs and contingencies specified by the client.

There are singular and multiple conditions generally placed on software evaluations that should be discussed. A software evaluation is usually a goal-oriented process, that is, determine the COTS software product that helps the system fulfill its requirements. Behind this process, however, a company or individual has conditions that must be met for an evaluation to be considered legitimate. For companies or individuals that have full discretion over funding, procurement and acquisition, the only condition required are the ones expressed in the requirements. Some companies and organizations, namely the US federal government, have additional conditions that are customarily placed on an evaluation. One of these conditions involves objectivity, that is, the evaluation should be conducted with fairness in an unbiased manner. Some government evaluations have resulted in large contracts being awarded to companies whose products are in alignment with the government's needs. This gives an eager or desperate company incentive to bias an evaluation in their favor. This biasing can take the form of manipulating a competitor's data, modifying their own data to appear more favorable, or even omitting unflattering data altogether. Biasing can also take the form of influencing the requirements process with the goal of revealing requirements that categorize their product as a sole source. To avoid these negative influences, the probabilistic evaluation process stresses the need to establish an independent software evaluator. The process also aids the client in addressing their needs and desires by establishing the attribute acceptance document without the (positive or negative) influence of the COTS software data. The conditions in this software evaluation process provide integrity, independence and objectivity for the client and the process.

IV. Conclusion

Companies and organizations are incorporating more and more COTS software products into their systems. In order to select an appropriate product, these organizations are utilizing various software evaluation processes. To use any evaluation technique, an evaluator needs a set of requirement specifications as a means of expressing the goals of the evaluation. Unfortunately, the standard requirements format does not work well for COTS software since the development and evolution of such products are controlled by individual product vendors. Clients, unfortunately, do not know how the COTS software will behave with no insight into the source code. Even with the standard format, bad and inadequate requirements have abounded resulting in cost overruns, schedule slips and lost profitability. These inadequacies stem from a lack of understanding between the client and the evaluator. This paper presents a new software requirements format called the attribute acceptance paradigm that is used in the context of a probabilistic COTS software evaluation process. This paradigm allows a client to utilize clearly defined attributes as well as technical performance trade-offs as an expression of his software needs. The paradigm includes descriptions for thresholds and contingencies to aid the software evaluator. The new acceptance paradigm requires the client to spend more time at the beginning of the evaluation process evaluating their needs and wants for the purpose of reducing the time involved in the standard iterative evaluation process.

References

[1]Morris, A. T., and Beling, P. A., "Extracting Acyclic Dependency Models from Quality Standards for COTS Software Evaluation ," *AIAA 1st Intelligent Systems Technical Conference*, Chicago, Illinois, Sep. 20-22, 2004.

[2]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.

[3]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.

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

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

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

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

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

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

[11]Swartout, W. R. and Balzer, R., "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM*, Vol. 25, No. 7, 1982, pp. 438-439.

[12]Hooks, Ivy F. and Farry, Kristin A., *Customer-Centered Products: Creating Successful Products Through Smart Requirements Management*, AMACOM, 2001.

[13]Dion, R., "Process improvement and the corporate balance sheet," *IEEE transactions on Software Engineering*, vol. 10, July 1993, pp. 283-285.

[14]Leffingwell, D., "Calculating the return on investment from more effective requirements management," *American Programmer*, Cutter Information Corp., Arlington, MA, April 1997, pp. 19-24.

[15]Gibson, J. E., *A System Analyst's Decalogue*, University of Virginia, July 1991.

[16]Sutcliffe, Alistair, "On the Inevitable Intertwining of Systems and Software Requirements," Department of Computation, University of Manchester, UK, April 2001.

[17]Elks, Carl R., and Johnson, Barry W., "Runtime Verification of COTS Components for Safety Related Systems," *AIAA 1st Intelligent Systems Technical Conference*, Chicago, Illinois, Sep. 20-22, 2004.

[18]Maiden, N.A.M., James, L., and Ncube, C., " *Evaluating Large COTS Software Packages: Why Requirements and Use Cases are Important*", in the Proceedings of the 1st International Workshop on Ensuring Successful COTS Development, Los Angeles, CA, May 1999.

[19]Nissen, Mark E., "Software Acquisition Top 10," Naval Postgraduate School, URL: http://web.nps.navy.mil/~menissen/mn3309/swtop10.htm [cited 14 September 2005].

[20]Popov, P., Riddle, S., Romanovsky, A. and Strigini, L., "On Systematic Design of Protectors for Employing OTS Items," *In Proceedings of the 27th Euromicro Conference*, Warsaw, Poland, 4-6 September 2001, pp. 22-29.

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

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

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

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