Defining Factors, Goals and Criteria for Reusable Component Evaluation

Presented at the CASCON '96 conference, Toronto, Canada, November 12-14, 1996.

Jyrki Kontio, Gianluigi Caldiera and Victor R. Basili
University of Maryland
Department of Computer Science
A.V.Williams Building
College Park, MD 20742, U.S.A.
Emails: [jkontio | caldiera | basili]@cs.umd.edu

Abstract:

This paper presents an approach for defining evaluation criteria for reusable software components. We introduce a taxonomy of factors that influence selection, describe each of them, and present a hierarchical decomposition method for deriving reuse goals from factors and formulating the goals into an evaluation criteria hierarchy. We present some highlights from two case studies in which the approach was applied. The approach presented in this paper is a part of the OTSO[1] method that has been developed for reusable component selection process.

1. Introduction

Software reuse is considered an important solution to many of the problems in software development. It is credited with improving the productivity and the quality of software development [1,4,15,24,30,33], and many organizations have claimed significant benefits from it [13,23].

Some organizations have implemented systematic reuse programs [13], which have resulted in in-house libraries of reusable components. Other organizations have supported their reuse with component-based technologies and tools. The increased commercial availability of embeddable software components, standardization of basic software environments (such as Microsoft Windows, Unix), and the explosive popularity of the Internet have resulted in a new situation for reusable software consumers: there are many more accessible reuse candidates. Consequently, many organizations are spending much time in reusable component selection since the choice of the appropriate components has a major impact on the project and resulting product.

Despite their importance, the issues and problems associated with the selection of suitable reusable components have rarely been addressed in the reuse community. Poulin et al. present an overall selection process [23] and include some general criteria for assessing the suitability of reuse candidates [32]. Some general criteria have been proposed to help in the search for potential reusable components [24,25]. Boloix and Robillard recently presented a general framework for assessing the software product, process and their impact on the organization [6]. However, none of this work is specific to off-the-shelf (OTS[2]) software selection, and the issue of how to define the evaluation criteria is not addressed. Furthermore, most of the reusable component literature does not seem to emphasize the sensitivity of such criteria to each situation.

We have developed a method that addresses the selection process of packaged, reusable software, or OTS as we refer to it in this paper. The method, called OTSO, supports the search,

[1] OTSO stands for Off-The-Shelf Option. The OTSO method represents a systematic approach to evaluate such an option.

[2] "OTS" stands for "off-the-shelf". The term originates from the term "COTS software", i.e., commercial off-the-shelf software. In this paper OTS refers to both commercial and in-house source code, executables, design, documentation, test cases, etc.

evaluation and selection of reusable software, and provides specific techniques for defining the evaluation criteria, comparing the costs and benefits of alternatives, and consolidating the evaluation results for decision making [18,19,21]. The main characteristics of the OTSO method are as follows:

- A defined, systematic process that covers the whole reusable component selection process.
- A method for estimating the relative effort or cost benefits of different alternatives.
- A method for comparing the "non-financial" aspects of alternatives, including situations involving multiple criteria.
- a predefined template for product quality characteristics to be tailored and used in each instance of the selection process.

Figure 1 shows the main activities in the OTSO reusable component selection process using a dataflow diagram notation. Each activity in presented as a process symbol – a circle – and artifacts produced or used are presented as data storage symbols in Figure 1. In the *search* phase, the goal is to identify potential candidates for further study. The *screening* phase selects the most promising candidates for detailed *evaluation*. In the *analysis* phase, the results of product evaluations are consolidated, and a decision about reuse is made. As the selected

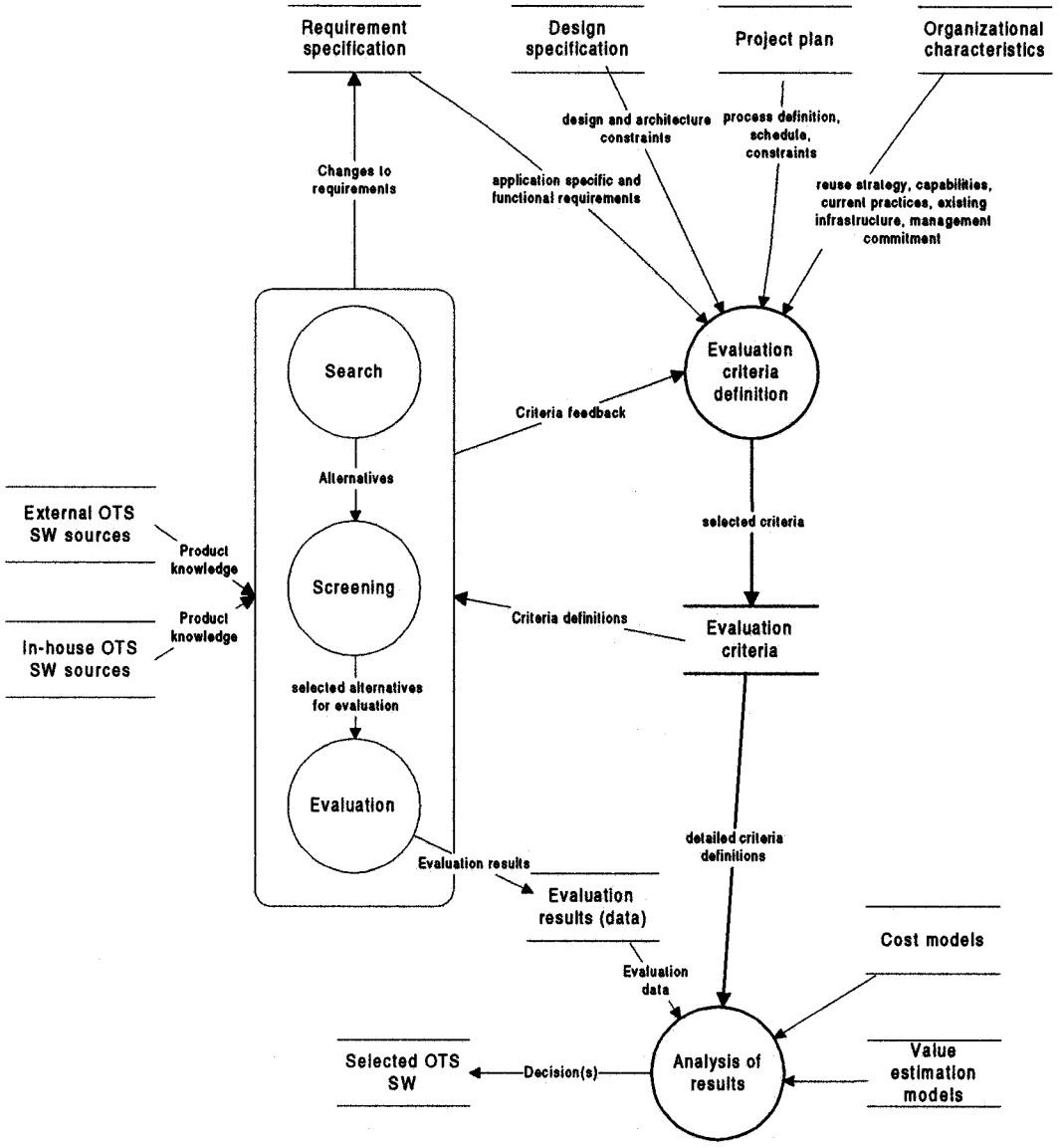

Figure 1: The main phases in the OTSO process

alternative is used (*deployed*), the effectiveness of the reuse decision, eventually, can be *assessed*. Reuse candidates are evaluated in different ways in all phases. The OTSO method is based on incremental, evolutionary definition and use of the evaluation criteria so that the criteria set can be gradually refined to support each phase. While Figure 1 presents the overall OTSO process, this paper presents the goal-driven criteria definition process of the method that has not been described publicly so far. Details about other aspects of the method are available in separate reports and publications [18,19,21].

The structure of this paper is as follows. Section 2 presents the factors that influence the OTS software selection and how the reuse goals can be formulated. Section 3 presents how the evaluation criteria can be defined and decomposed. Section 4 presents applicable results from the two case studies with the OTSO method. Finally, the conclusion section discusses the relevance of our results.

2. Factors in Reusable Software Selection

The overall relationships among influencing factors, reuse goals and evaluation criteria are presented in Figure 2. The first main task in reusable software evaluation is to define the reuse goals. This must be based on a careful analysis of the influencing factors. We identified five groups of factors that primarily influence the OTS software selection. In the following sections we discuss each of these groups.

Application requirements are likely to be the most important factor in evaluating reusable software. Such requirements can include functional requirements (such as the ability to manage and display graphical geographical data) and non-functional requirements (such as available memory or speed of operations). The requirement specification, if available, should be used as a basis for interpreting such requirements.

The requirement specification, however, can only give partial support for the interpretation of software reuse goals for two reasons. First, the requirement specification typically does not define how the system should be implemented or what components could be implemented through OTS software. Considering the use of OTS software in a system means making some assumptions about the architecture of the system, and requires some decisions on which system features should be covered by the OTS software. Second, the requirement specification may not be detailed enough for evaluating OTS software alternatives. In both of these cases the formulation of software reuse goals requires interpretation and further refinement of requirements, and some design concepts.

Application and domain architecture introduce additional elements that need to be evaluated. The architecture, in this context, provides a set of constraints deriving from how particular applications are built: this includes, for instance, components and design patterns used or assumed, communication and interface standards, platform characteristics. All of this introduces a set of constraints that may make integration of some alternatives impractical or costly. Some kind of application mediator, or "middleware", may be used to overcome such problems. This mediator, however, needs to be either developed or acquired from another source and this provides, for instance, a cost increment that needs to be estimated.

The application domain may also have some specific characteristics that are not addressed by OTS software developed for other domains (for example, real-time applications vs. batch processing). Sometimes the architecture is a given and acts as a constraint in the OTS software selection; sometimes the selection of the right OTS software may determine or influence the system architecture.

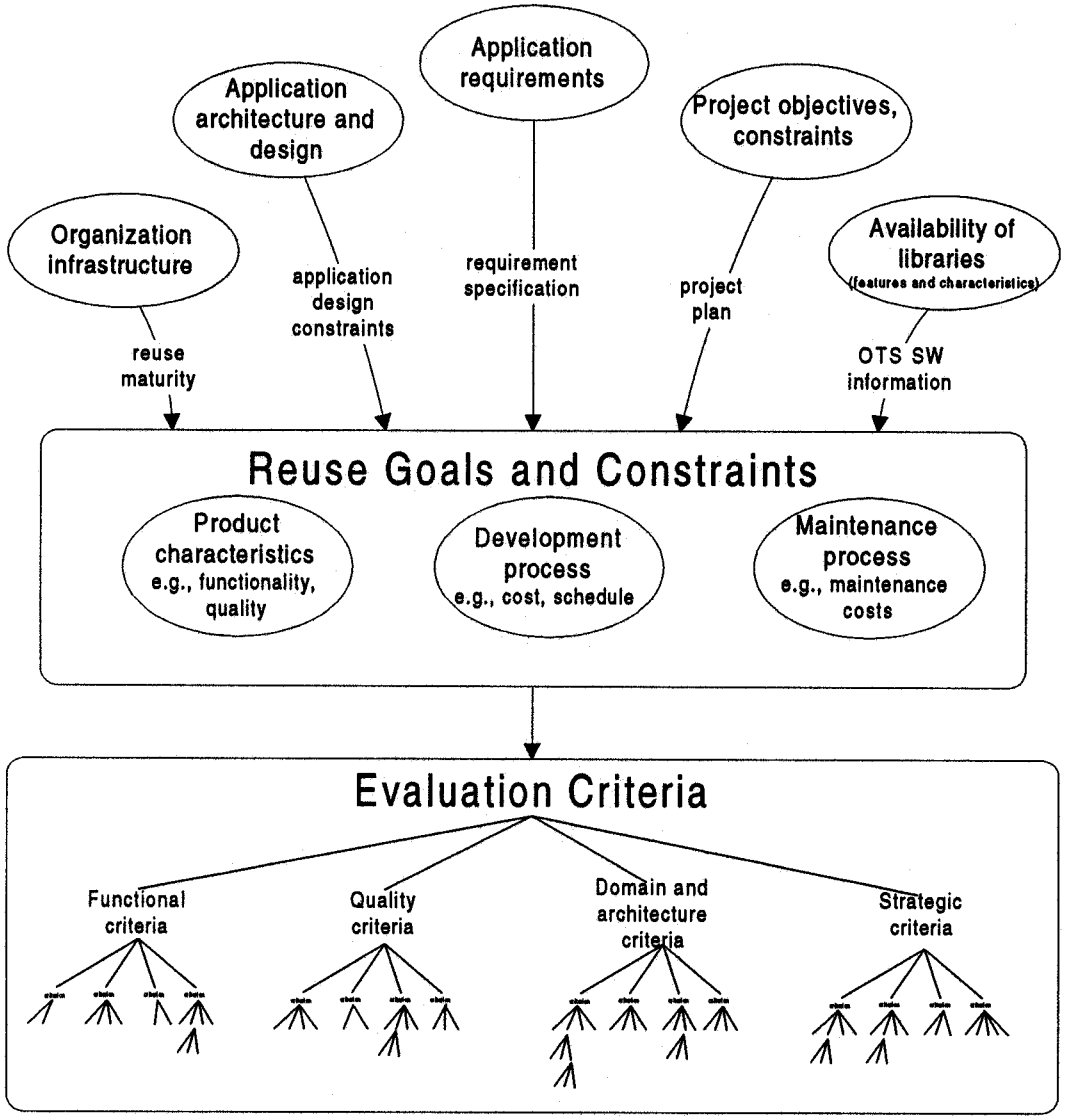

Figure 2: Factors influencing the selection of reusable off-the-shelf software

Project objectives and constraints may influence the library selection through the schedule or the budget of the project. For instance, early deadlines or low personnel budget[3] may require the use of externally produced software. Project objectives may also imply the use of external software if, for example, these external libraries may provide a better way to comply with standards or may be proven reliable in implementing some aspects of the library. There may also be some organizational constraints that are set for the project, such as availability of personnel with specific skills.

The availability of features in software reuse candidates also affects the evaluation criteria definition. This works in two ways. On one hand, it is important to check that the evaluation criteria are based on realistic expectations. That is, the criteria set should not assume characteristics that are not provided by any OTS software alternative. On the other hand, it may be useful to know about the possibilities that OTS

[3] Note that this example is not meant to imply that the use of OTS software necessarily results in lower overall costs. The example highlights the usual situation where the use of OTS software changes the cost structure of a project, e.g., development costs may be lowered but software acquisition costs are higher.

software alternatives offer but that may not have been included in the requirement specification.

Finally, an organization's reuse infrastructure and reuse maturity should also be considered when defining reuse goals. Reuse maturity comprises several issues: an organization's experience in reuse, its commitment and interest in continuing systematic reuse, the knowledge and skills of personnel responsible for reuse, the availability of specific tools for supporting reuse (configuration management tools, information databases, etc.), and the existing software development environment [9,17]. Reuse maturity is particularly important for the in-house production of reusable components. Also, if an organization has no experience in OTS software reuse, it may have a limited ability to integrate OTS software and to estimate the effort required for OTS software integration.

The main point of this discussion is that the *evaluation criteria should be developed with full awareness of all these factors.* In most cases, this requires that each factor be explicitly analyzed, and documented and used as input in the final definition of the evaluation criteria.

Each OTS software reuse situation is different, and so are the reuse goals associated with it. Based on the analysis of factors as described in the previous section, the reuse goals for the project need to be stated explicitly. The OTS software reuse goal statement essentially should describe the following:

- Where and how OTS software is to be used in the application;
- The expected benefits of OTS software reuse, such as functionality, quality, schedule impact, or effort savings;
- Possible constraints for OTS software reuse;
- Cost budget for the use of OTS software.

The reuse goal statement should be documented explicitly, although initially the goal statement may seem abstract and simple. Our experience indicates that it will be revised and become more detailed as the OTS evaluation progresses.

Reuse objectives can be divided into development process goals, maintenance process goals and product characteristics goals. Development process goals relate to the cost, effort and schedule of the development project. The maintenance process goals deal with issues such as the ease or cost of maintenance and who will be responsible for maintenance. Product characteristics goals refer to product functionality and product quality.

3. Evaluation Criteria

3.1 Classes of evaluation criteria

The factors and goals described in Figure 2 determine the reuse goals for the system. The content and priorities of these goals determine which characteristics must be considered in the of the OTS software selection process. The evaluation criteria themselves can be categorized into four main areas: (i) functional requirements, (ii) product quality characteristics, (iii) strategic concerns, and (iv) domain and architecture compatibility.

Functional requirements: These refer to identifiable, functional features or characteristics that are specific to the particular situation. These criteria are derived from the requirement or design specification and are expressed in the form of requirements. Here are two examples from an application dealing with geographical data:

- Display ocean bathymetry data
- Show political boundaries.

Product quality characteristics are common to a broader set of reuse situations. Typically the structure and relationships of these characteristics remain the same but their acceptable values may vary from case to case. Three examples are:

- Defect rate
- Compliance to the project user interface guidelines
- Clarity of documentation.

Strategic concerns: These are the short-term and the long-term effects of the reuse candidate, the cost-benefit issues and the organizational issues beyond the scope of the project in question. These help to consolidate information for decision making. Three examples are:

- Acquisition costs
- Effort saved
- Vendor's future plans.

Domain and architecture compatibility: An application domain or the software architecture

Attribute of GQM	Explanation	Examples	
Object (entity)	The entity being analyzed, e.g., OTS product, OTS vendor	OTS product	OTS vendor
Focus (issue)	The attributes that are of interest, e.g, cost, reliability, or efficiency.	Cost	Viability
Purpose	Evaluate: evaluate the characteristics of the entity w.r.t. a relevant benchmark. This attribute is typically the same in all reuse evaluation cases.	Evaluate	Evaluate
Point of view (perspective)	Whose interest is being expressed, e.g., project manager, corporation, customer, developer, etc.	Customer	Development organization

Table 1: GQM-based evaluation criteria definition template

may also require specific characteristics from reuse candidates. For instance, all flight-control software must be very reliable and must be developed with time-sensitive and reactive issues in mind. A reuse candidate originally developed for accounting software may have fundamental design and performance characteristics that make it unsuitable for such an application area.

Domain compatibility refers to how well the reuse candidate and its features map into the domain terminology and concepts. In the case of object oriented reuse candidates, this can refer to a match between domain objects and object definitions in the reuse candidate. Architecture compatibility refers to software or hardware architecture requirements that are common to the application area.

Examples are listed below:

Domain compatibility:
- system states can be modeled and represented
- geographical data manipulation capability

Architecture compatibility:
- supports or is compatible with CORBA
- compatible with Microsoft Windows OLE.

The evaluation criteria must be customized for each selection situation. The functional requirements, which often are central to the selection process, are often unique to each application. For the product quality characteristics and strategic concerns, it is possible to define some templates that have stable elements accross applications. As an input to product quality characteristics, there are several possible sources [5,7,10,16]. Figure 3 shows an example of the product quality factors we defined for one of our case study projects.

3.2 Hierarchical decomposition of evaluation criteria

The evaluation criteria are derived from the factors and goals discussed in the previous sections. The first step in this process is to define the *evaluation goals* using the GQM approach, as it provides a well-defined template for documenting such evaluation goals [2,3]. Table 1 presents the template used for GQM goals. The *object* attribute and *focus* attributes can be derived often directly from reuse goals. For example, if a reuse goal is to reduce development cycle time, we are evaluating the process (object) and its duration (focus).

The *purpose* attribute can range from simple characterization to understanding, evaluation and even prediction [2]. However, most often the purpose is evaluation. The point of view attribute is relevant when there are different stakeholders interested in the results and their views need to be considered. For example, developer and user perspectives may be different in terms of required functionality of the product.)

The basic steps of criteria decomposition are the following [18]:

1. Identify and formulate evaluation goals using the template given in Table 1. For example, an evaluation goal could be stated as follows: object/entity:
2. For each evaluation goal, define a set of high-level criteria or questions that characterize it.
3. For each criterion, write down an unambiguous definition of it.
4. If the value for the criterion can be determined with an objective measurement, observation or judgment, call it an evaluation attribute, and

Type of Evaluation Attribute	Examples
Measurement	• memory used when loaded • time to initialize • number of bugs found during evaluation • support of incremental image loading
Description	• list of reported bugs • description of the "undo" function
Subjective assessment	• estimate of vendor's growth potential • look and feel of GUI

Table 2: Examples of evaluation attributes

continue to decompose and define other criteria. If the criterion is too abstract to be measured with a single metric, if it has too many aspects to be assessed through observation or if it cannot be judged objectively, continue decomposing it.

The number of items at each level should be less than 10, preferably around 3 to 5.

The objective of the criteria-definition process is to decompose criteria into a set of concrete, measurable, observable or testable *evaluation attributes*. An evaluation attribute can be an observation, a measurement, or a piece of information to be obtained. Table 2 lists examples of possible types of evaluation attributes.

Once the criteria have been defined, the OTSO method relies on the use of the Analytic Hierarchy Process (AHP) for consolidating the evaluation data for decision-making purposes. The AHP technique was developed by Thomas Saaty for multiple-criteria decision-making situations [26,27]. The technique has been widely and successfully used in several fields [28], including software engineering [11] and software selection [14,22]. It has been reported effective in several case studies and experiments [8,12,28,31]. Due to the hierarchical treatment of our criteria, AHP fits well into our evaluation process as well. AHP is supported by a commercial tool that supports the entering of judgments and performs all the necessary calculations [29].

The AHP is based on the idea of decomposing a multiple-criteria decision-making problem into a criteria hierarchy. At each level in the hierarchy the relative importance of factors is assessed by comparing them in pairs. Finally, the alternatives are compared in pairs with respect to the criteria. Rephrasing the AHP approach in the OTSO framework, the evaluation proceeds as follows [11,27]:

1. Define the importance of factors on each level.
2. Define the preferences of alternatives over the lowest level factors in the criteria tree.
3. Check the consistency of rankings and revise them if necessary.
4. Present the results of the evaluation, the alternative with the highest priority being the one that is recommended as the best alternative.

The rankings obtained through paired comparisons between the alternatives are converted to normalized rankings using the "eigenvalue" method, preferably using a software tool that automates the calculation process [27].

This process can be illustrated with a simple example. Assume that one needs to decide which Web browser to use, Internet Explorer or Netscape (alternatives). Assume that the evaluation criteria decomposition process has resulted in just criteria, price and popularity. According to the AHP method we would first determine the relative importance of factors, resulting in weights for each. New both alternatives (Internet Explorer and Netscape) are compared against these two criteria and their relative rankings (weights) are obtained. Based on this information, the relative preferences of alternatives can be calculated and expressed as numbers totaling one. More information about the details of the AHP method or the Expert Choice tool is available separate publications [26,27,29].

From our perspective, the main advantage of AHP is that it provides a systematic, validated approach for consolidating information about alternatives using multiple criteria. AHP can be used to "add up" the characteristics of each alternative. Furthermore, an additional benefit of AHP is that we can choose the level of consolidation. We recommend that consolidation be carried out only to a level that is possible without sacrificing important information. On the other hand, some consolidation may avoid

overwhelming the decision makers with too much detailed, unstructured information.

The weighting of alternatives is done using the AHP method, preferably using a supporting tool [29]. Preferences are collected and consolidated to the level stakeholders prefer. The AHP allows the consolidation of all qualitative information and financial information into a single ranking of alternatives. However, we believe that this would condense valuable information too much. Instead, we recommend that information about the evaluation be consolidated to a level where a few main items remain so that stakeholders can discuss their impact and preferences. The full consolidation can be done at the end as a sanity check, if desired.

4. Case Studies

We carried out two case studies using the OTSO method. The results of these case studies are reported separately [18,19,21]. The first case study assessed the overall feasibility of the method and the second one focused on the comparison of analysis methods. Both case studies took place in the NASA's Earth Orbiting System (EOS) program with Hughes Information Technology Corporation and were dealing with real software development projects facing a COTS selection problem.

Our first case study dealt with the selection of a library that would be used to develop an interactive, graphical user interface for entering location information on Earth's surface areas. This case study used the OTSO method's hierarchical and detailed criteria definition approach. Part of the criteria hierarchy is presented in Figure 3. The main conclusion was that the OTSO method was a feasible approach in COTS selection and its overhead costs were marginal [18].

The first case study also showed that OTS package features can change the application requirements: one of the OTS alternatives was able to display ocean bathymetry data graphically. Although this was not initially specified as a requirement, the application designers considered it a valuable feature and proposed it to be included in the requirements specification. This important feedback loop is characterized by the arrow from the search/screening/evaluation contour in Figure 1.

The second case study dealt with the selection of a hypertext browser for the EOS information service. This case study included a comparison between two analysis methods, the AHP method and a weighted scoring method.

A total of over 48 tools were found during the search for possible tools. Based on the screening criteria, four of them were selected for hands on evaluation. The evaluation criteria were derived from existing, broad requirements. However, as in the first case study, the requirements had to be elaborated and detailed substantially during this process.

This case study further supported our conclusion of the low overhead of the OTSO method. Furthermore, this case study involved several evaluators, and our criteria definition approach improved the efficiency and consistency of the evaluation. We also found an unexpected result when comparing the two analysis methods: they yielded different rankings of the COTS alternatives even though they were based on the same data [19,20]. In our opinion, this highlights the importance of appropriate analysis and data consolidation techniques in such evaluations.

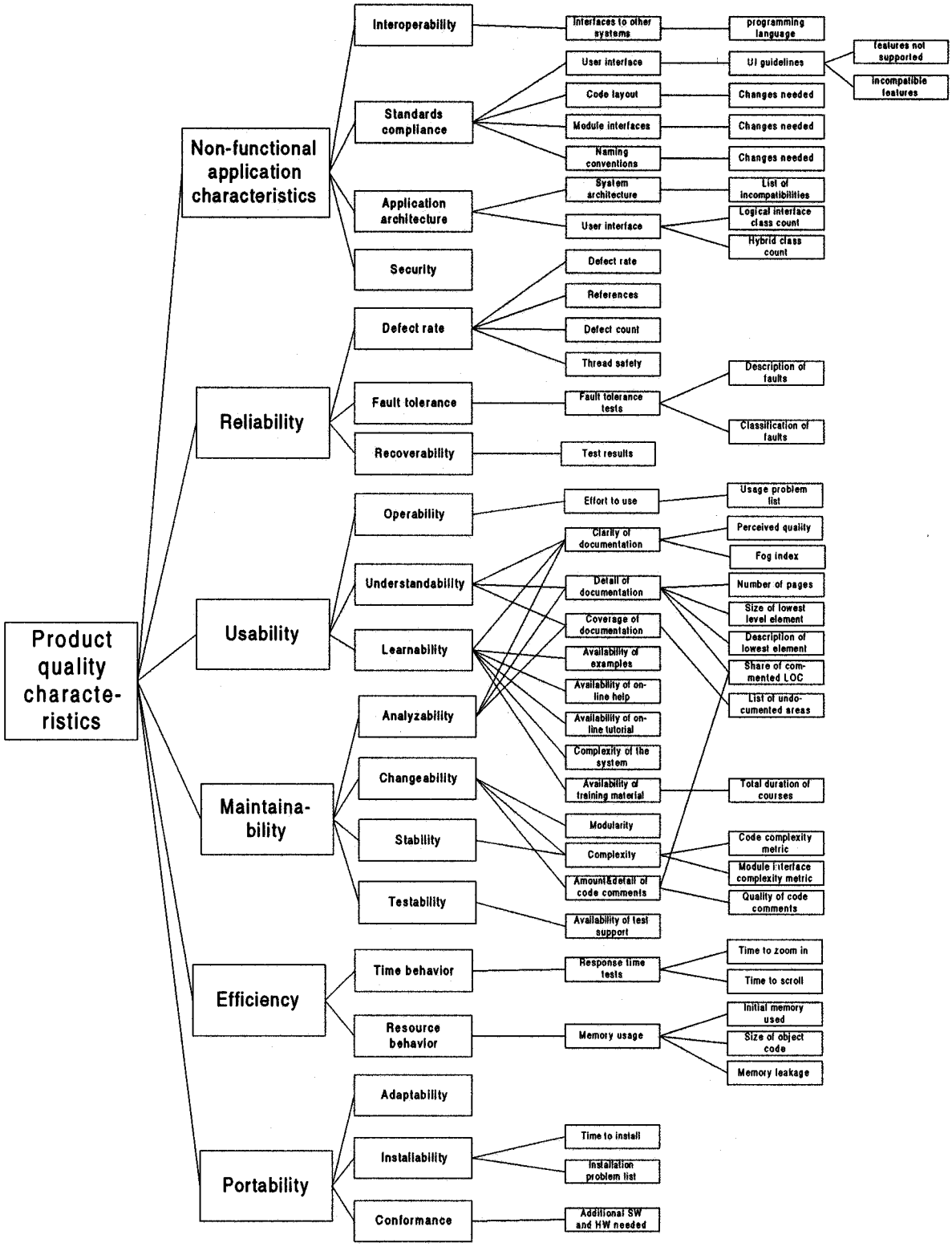

Figure 3: Example of a product evaluation criteria hierarchy

5. Conclusions

The OTSO method was developed to consolidate some of the best practices we have been able to identify for OTS software selection. The experiences from our case studies indicate that our method is feasible in an operational context: it improves the efficiency and consistency of evaluations, it has low overhead costs, and it makes the COTS selection decision rationale explicit in the organization. The detailed evaluation criteria also contribute to the refinement of application requirements.

The evaluation criteria definition approach presented in this paper is a central element of the OTSO method. The underlying assumption of our approach is that as each situation is different, the factors, goals and evaluation criteria will need to be defined for each situation separately. By formalizing this criteria definition process, it is possible to reuse the OTS software selection experiences better, leading to a more efficient and reliable selection process.

Although our case studies were both performed in the same application domain, we have not encountered any domain specific characteristics that would limit the applicability of the method in other domains. Also, while the case studies themselves were relatively small, the evaluation processes, and the resulting criteria, were quite extensive. This leads us to suggest that the method may be able to scale up to larger situations as well. However, further validation is necessary to determine this with more confidence.

6. Acknowledgments

This work has been supported by the Hughes Information Technology Corporation and the EOS Program, as well as by the Software Engineering Laboratory, a joint software process improvement organization between NASA, Computer Sciences Corporation and University of Maryland.

7. About the Authors

Jyrki Kontio is a researcher at the Department of Computer Science at University of Maryland. His research interests include risk management, process improvement, process modeling, technology management, and software reuse. He is on a leave-of-absence from Nokia Research Center, completing his Ph.D. on software risk management. He can be reached at jkontio@cs.umd.edu.

Gianluigi Caldiera is a Research Manager at the Department of Computer Science at University of Maryland. His research and professional activities are in software engineering, with special focus on software quality assurance and management, software metrics, software reuse and software factories, software process improvement, and quality standards. In his more than 15 years of professional and academic experience, he has consulted for Government and industry in both Europe and United States. He can be reached at gcaldiera@cs.umd.edu.

Victor R. Basili is a professor in the Institute for Advanced Computer Studies and the Department of Computer Science. He is a co-founder and a director of the Software Engineering Laboratory. Professor Basili is also co-editor-in-chief of the International Journal of Empirical Software Engineering. He can be reached at basili@cs.umd.edu.

8. References

[1] B. Barnes, T. Durek, J. Gaffney, and A. Pyster. "A Framework and Economic Foundation for Software Reuse,". In: *Tutorial: Software Reuse: Emerging Technology*, ed. W. Tracz. Washington: IEEE Computer Society, 1988.pp. 77-88.

[2] V. R. Basili, "Software Modeling and Measurement: The Goal/Question/Metric Paradigm," CS-TR-2956, 1992. Computer Science Technical Report Series. University of Maryland. College Park, MD.

[3] V. R. Basili, G. Caldiera, and H. D. Rombach. "Goal Question Metric Paradigm,". In: *Encyclopedia of Software Engineering*, ed. J. J. Marciniak. New York: John Wiley & Sons, 1994.pp. 528-532.

[4] T. Birgerstaff and C. Richter, "Reusability Framework, Assessment, and Directions," *IEEE Software*, vol. 4, March. pp. 41-49, 1987.

[5] B. W. Boehm, J. R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," pp. 592-605, 1976. Pcodeedings of the Second International Conference on Software Engineering. IEEE.

[6] G. Boloix and P. N. Robillard, "A Software System Evaluation Framework," *IEEE Computer*, vol. 28, 12. pp. 17-26, 1995.

[7] J. P. Cavano and J. A. McCall, "A Framework for the Measurement of Software Quality," *ACM SIGSOFT Software Engineering Notes*, vol. 3, 5. pp. 133-139, 1978.

[8] A. T. W. Chu and R. E. Kalaba, "A Comparison of Two Methods for Determining the Weights Belonging to Fuzzy Sets," Journal of Optimization Theory and Applications, vol. 27, 4. pp. 531-538, 1979.

[9] T. Davis, "Toward a reuse maturity model," eds. M. L. Griss and L. Latour. pp. Davis_t-1-7, 1992. Proceedings of the 5th Annual Workshop on Software Reuse. University of Maine.

[10] M. S. Deutsch and R. R. Willis. "*Software Quality Engineering - A total Technical and Management Approach*," Englewood Cliffs: Prentice-Hall, 1988. 317 pages.

[11] G. R. Finnie, G. E. Wittig, and D. I. Petkov, "Prioritizing Software Development Productivity Factors Using the Analytic Hierarchy Process," *Journal of Systems and Software*, vol. 22, pp. 129-139, 1995.

[12] E. H. Forman, "Facts and Fictions about the Analytic Hierarchy Process," *Mathematical and Computer Modelling*, vol. 17, 4-5. pp. 19-26, 1993.

[13] M. L. Griss, "Software reuse: From library to factory," *IBM Systems Journal*, vol. 32, 4. pp. 548-566, 1993.

[14] S. Hong and R. Nigam. "Analytic Hierarchy Process Applied to Evaluation of Financial Modeling Software,". In: *Proceedings of the 1st International Conference on Decision Support Systems, Atlanta, GA*, Anonymous1981.

[15] J. W. Hooper and R. O. Chester. "*Software Reuse: Guidelines and Methods*," R.A. Demillo (Ed). New York: Plenum Press, 1991.

[16] ISO. "*Information technology - Software product evaluation - Quality characteristics and quidelines for their use, ISO/IEC 9126:1991(E)*," Geneve, Switzerland: International Standards Organization, 1991.

[17] P. Koltun and A. Hudson, "A Reuse Maturity Model," ed. W. B. Frakes. pp. 1-4, 1991. Proceedings of the 4th Annual Workshop on Software Reuse. University of Maine. Department of Computer Science.

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

[19] J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection," 1996. Proceedings of the 18th International Conference on Software Engineering.

[20] J. Kontio and S. Chen, "Hypertext Document Viewing Tool Trade Study: Summary of Evaluation Results," 441-TP-002-001, 1995. ECS project Technical Paper. Hughes Corporation, ECS project.

[21] J. Kontio, S. Chen, K. Limperos, R. Tesoriero, G. Caldiera, and M. S. Deutsch, "A COTS Selection Method and Experiences of Its Use," 1995. Proceedings of the 20th Annual Software Engineering Workshop. NASA. Greenbelt, Maryland.

[22] H. Min, "Selection of Software: The Analytic Hierarchy Process," *International Journal of Physical Distribution & Logistics Management*, vol. 22, 1. pp. 42-52, 1992.

[23] J. S. Poulin, J. M. Caruso, and D. R. Hancock, "The business case for software reuse," *IBM Systems Journal*, vol. 32, 4. pp. 567-594, 1993.

[24] R. Prieto-Díaz, "Implementing faceted classification for software reuse," *Communications of the ACM*, vol. 34, 5.1991.

[25] C. V. Ramamoorthy, V. Garg, and A. Prakash, "Support for Reusability in Genesis," pp. 299-305, 1986. Proceedings of Compsac 86. Chicago.

[26] T. L. Saaty. "*Decision Making for Leaders*," Belmont, California: Lifetime Learning Publications, 1982. 291 pages.

[27] T. L. Saaty. "*The Analytic Hierarchy Process*," New York: McGraw-Hill, 1990. 287 pages.

[28] T. L. Saaty. "Analytic Hierarchy,". In: *Encyclopedia of Science & Technology*, Anonymous McGraw-Hill, 1992.pp. 559-563.

[29] T. L. Saaty, Expert Choice software 1995, ver. 9, rel. 1995. Expert Choice Inc. IBM. Windows 95.

[30] W. Schäfer, R. Prieto-Díaz, and M. Matsumoto. "*Software Reusability*," W. Schäfer, R. Prieto-Díaz, and M. Matsumoto (Eds). Hemel Hempstead: Ellis Horwood, 1994.

[31] P. J. Schoemaker and C. C. Waid, "An Experimental Comparison of Different Approaches to Determining Weights in Additive Utility Models," *Management Science*, vol. 28, 2. pp. 182-196, 1982.

[32] W. Tracz, "Reusability Comes of Age," *IEEE Software*, vol. 4, July. pp. 6-8, 1987.

[33] W. Tracz. "Software Reuse: Motivators and Inhibitors,". In: *Tutorial: Software Reuse: Emerging Technology*, ed. W. Tracz. Washington: IEEE Computer Society, 1988.pp. 62-67.

Note: Some of the technical reports and papers describing the OTSO method are available through "http://www.cs.umd.edu/users/jkontio/".