RL-TR-93-80
Final Technical Report
May 1993

CONFLICT RESOLUTION (CORE) FOR SOFTWARE QUALITY FACTORS

Rochester Institute of Technology

Jeffrey A. Lasky, Kevin H. Donaghy

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

Rome Laboratory
Air Force Materiel Command
Griffiss Air Force Base, New York

This report has been reviewed by the Rome Laboratory Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RL-TR-93-80 has been reviewed and is approved for publication.

APPROVED: Roger J. Dziegiel, Jr.

ROGER J. DZIEGIEL, JR.
Project Engineer

FOR THE COMMANDER: John A. Graniero

JOHN A. GRANIERO
Chief Scientist
Command, Control & Communications Directorate

If your address has changed or if you wish to be removed from the Rome Laboratory mailing list, or if the addressee is no longer employed by your organization, please notify RL (C3CB) Griffiss AFB NY 13441-5700. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1993	Final May 90 - Sep 92

4. TITLE AND SUBTITLE
CONFLICT RESOLUTION (CORE) FOR SOFTWARE QUALITY FACTORS

5. FUNDING NUMBERS
C - F30602-88-D-0026
Task 36
PE - 62702F
PR - 5581
TA - 20
WU - PB

6. AUTHOR(S)
Jeffrey A. Lasky, Kevin H. Donaghy

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Rochester Institute of Technology
1 Lomb Drive
Rochester NY 14613-5700

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Rome Laboratory (C3CB)
525 Brooks Road
Griffiss AFB NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
RL-TR-93-80

11. SUPPLEMENTARY NOTES
Rome Laboratory Project Engineer: Roger J. Dziegiel, Jr./C3CB (315) 330-2054

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

The software quality problem can be formulated as maximizing the quality goals within the constraints of cost, schedule and technical feasibility. This is a difficult problem which belongs to a class known as multiobjective or multiattribute optimization problems. These problems are characterized by the presence of multiple, conflicting goals accompanied by a large candidate solution space. The goals conflict because they are somehow interrelated.

In software development, there are several quality characteristics, or software quality factors, that inherently conflict. For example, efficiency and maintainability conflict and the objective is to improve code understandability, since efficiency frequently required reliance on exceptional code. The same is true for expandability and reliability (increased risk to acquire more functionality), safety and availability (fail-soft/fail-safe requirements reduce the set of available system capabilities).

This effort developed a prototype tool which provides computer support for the Rome Laboratory Software Quality Methodology. Conflict Resolution (CORE) determines whether quality factor goals are achievable. If not, CORE then adjusts factor goals until an achievable solution is identified. In a session, there may be several candidate solutions.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Software Wuality, Software Quality Specification, Metrics	52
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	U/L

TABLE OF CONTENTS

SECTION I

EXECUTIVE SUMMARY

1 INTRODUCTION

This technical report represents the results of the System Quality Attributes Study. This work was performed for Rome Laboratory (RL) by Rochester Institute of Technology (RIT) under contract no. F30602-88-D-0026.

2 BACKGROUND

This effort continues work related to the software quality research program sponsored by Rome Laboratory (formerly Rome Air Development Center) since 1976. RIT's prior contract results are reported in Software Quality Measurement Methodology Enhancements Study Results [1] and in Software Quality Methodology Study Results [2].

3 PRESENT STUDY

The objective of the present study is to extend the results of prior work. One of our study's principal aims was to define a software development process architecture from a software quality perspective.

The other principal aim was to implement a proof-of-concept tool designed to support the resolution of software quality factor conflicts. The current implementation of the Assistant for the Specification of Quality Software [3] does not completely address the quality factor conflict problem.

4 CONTRACT RESULTS

4.1 Quality driven process architecture

We have defined a quality driven software development process architecture. The novel feature of the architecture is that it explicitly links software product quality goals to the software development process. We have thus potentially supplied an answer to a question of current intense interest: how should the software development process be defined in order to meet specific software product quality requirements?

The basic elements of the architecture are:

1. A software quality domain analysis.
2. A method for deriving quality requirements from the domain analysis.
3. A method for validating the derived quality requirements against technical feasibility constraints.
4. A method for determining whether the validated quality requirements can be achieved given cost and schedule constraints.

5. A process generator that produces a development process to meet the validated quality requirements. The development process is composed of an ordered sequence of development tasks, or subprocesses

6. A set of process validations specific to each development subprocess.

Since the development process has been derived by product quality requirements, and the derived process measures are designed to validate correct process execution, compliance with the measures is equivalent to compliance with the product quality requirements. Such compliance represents the link between software product quality requirements and the software development process itself.

4.2 Resolution of quality factor conflicts

We have implemented CORE (COnflict REsolution), a working prototype tool which provides computer support to Rome Laboratory's Software Quality Methodology [4]. The problem which we have initially addressed is to determine if a set of software quality factor goals is technically achievable, given the factor and criteria interrelationships specified in Volume II of [4]. If CORE determines that the quality factor goals are not achievable, CORE then adjusts factor goals until an achievable solution (a feasible set of quality goals) is identified. In practice, CORE typically finds several candidate solutions.

This quality factor problem belongs to a large class of problems collectively known as multiobjective or multiattribute optimization problems. These problems are characterized by the presence of multiple, conflicting goals accompanied by a large candidate solution space. The goals conflict because they are somehow interrelated. Thus, another characteristic of this problem class is that attempts to improve one objective, by reallocating resources from a common resource pool, will usually degrade some other interrelated objective(s).

In software development, there are several software quality factors that inherently conflict. For example, efficiency and maintainability conflict if the objective is to improve code understandability, since efficiency frequently requires reliance on exceptional code.

We successfully applied CORE to the Surveillance and Identification function of the Airborne Radar System example used in Volume II of [4]. A sample session which illustrates the capabilities of CORE is included in section III of this report.

SECTION II

QUALITY DRIVEN SOFTWARE DEVELOPMENT PROCESS ARCHITECTURE

1 INTRODUCTION

Over the past several years, there has been a dramatic increase in studying and improving the software development process. The hypothesis underlying the interest in process is that improvements in the software development process will translate into higher quality software products. Based on field experiences reported to date, it seems likely that the hypothesis generally will be demonstrated true. To date, the evidence has, for the most part, been a reduction in the number of field failures experienced by users. However, a corollary question will be of more immediate interest to those responsible for planning and managing software development projects: how to design a software development process to meet the full spectrum of potential product quality goals.

This section describes a quality driven software development process architecture which explicitly links software product quality goals to the software development process. Several elements of Rome Laboratory's long-standing software quality research program are incorporated in the framework.

2 PROCESS ARCHITECTURE

2.1 Basic elements

The basic elements of the architecture are:

1. A software quality domain analysis.
2. A method for deriving quality requirements from the domain analysis.
3. A method for validating the derived quality requirements against technical feasibility constraints.
4. A method for determining whether the validated quality requirements can be achieved given cost and schedule constraints.
5. A process generator that produces a development process to meet the validated quality requirements. The development process is composed of an ordered sequence of development tasks, or subprocesses
6. A set of process validations specific to each development subprocess.

Since the development process has been derived by product quality requirements, and the derived process measures are designed to validate correct process execution, compliance with the measures is equivalent to compliance with the product quality requirements. Such compliance represents the link between software product quality requirements and the software development process itself.

2.2 Domain Analysis

A domain analysis can be viewed as systems analysis applied to a family of related programs (the domain). A domain model is the primary output of a domain analysis. The model is an abstract representation of the domain's logical and physical requirements. Initially, domain analysis was undertaken in order to identify commonalties of functions, designs and source code. The results formed the basis for libraries containing reusable software components. Current perspective is that domain analysis should provide the foundation for the reuse of software development products from all phases of the development life-cycle [5].

Rome Laboratory applied the domain analysis concept in order to determine the software quality requirements for five Air Force mission areas. The most detailed quality requirements were derived for the satellite mission area. Although the analysis was difficult and time-consuming, the results suggest that a software quality requirements domain analysis provides deep insights into the domain's generic quality characteristics [6].

2.3 Specification of quality requirements

Using the software quality domain analysis as a baseline, the quality requirements for a specific proposed system in the domain are then derived. Rome Laboratory developed an expert system, the Assistant for the Specification of Software Quality (ASQS) [3] to support the activity of determining software quality requirements. This activity can also be carried out manually, although the complexity of the task is considerable. Information from several sources is combined with the baseline domain analysis to produce an initial quality requirements specification. The quality requirements are stated in terms of a quantitative goal for each key software quality factor [4].

2.4 Technical feasibility analysis

Software quality factors are not technically independent. In some cases, it is presumed not feasible to achieve, for general design and technical reasons, simultaneously high levels of quality for several quality factors in the same product. Thus, it is necessary to subject the software quality requirements specification to a feasibility analysis.

In section III, Resolution of Quality Factor Conflicts, we describe an initial implementation of CORE (Conflict Resolution), a tool that checks the initial quality specification for technical feasibility. If the results suggest that the specification is not achievable according to the sets of interrelationship rules, then the tool lowers the values of certain quality factors until a feasible set(s) of quality specifications are identified. Based on the alternatives generated by the tool, the initial quality specification is then modified.

2.5 Cost/schedule feasibility analysis

Since resources are always limited, it may not be feasible to achieve quality requirements. Thus, it is necessary to analyze the current quality specification in light of cost and schedule constraints. Since software development projects are notoriously non-linear in use of time and resources, the method of feasibility analysis should be able to capture the dynamics of software development projects.

The systems dynamics modeling approach has been successfully applied to a NASA [7] and later to a USAF software development project [8]. The costs and schedule requirements of development technologies related to software quality objectives are determinable, with

research, for individual quality technologies. The existing models can then be modified to incorporate the inclusion of these additional project factors. A simulation will then show the impacts of the quality technologies on overall project cost and schedule. If the results show that cost and schedule constraints are being violated, then the quality specification can be further modified until an acceptable tradeoff between quality, cost and schedule is identified.

2.6 Process generation

Once a final software quality requirements specification has been created, then conceptually a development process can be automatically generated from a database of process fragments. The generated process, then, will directly reflect the software's quality requirements. Initial work on identifying and defining process fragments has been conducted under the sponsorship of IEEE Working Group P-1074. Although this particular set of process fragments does not generally include the scope of quality technology fragments envisioned here, it does represent an important starting point. A subset of these fragments has already been incorporated in a process modeling tool [9]. Process modeling in general is a very active area within the software engineering community [10]. A detailed process description of one software quality technology, Cleanroom Engineering, has recently become available [11].

For use in the type of process generation envisioned here, each quality technology process fragment definition would include normalized standard cost and standard schedule resource requirements. The most likely normalizations would be software size or function points. The same normalized cost and schedule data are used by the cost/schedule feasibility analysis described in section 2.5 above.

2.7 Process validation

The validations are expressed in the ETVX *paradigm* [12]. An ETVX description for each development subprocess, or activity, is composed of activity ENTRY criteria, TASK descriptions, VALIDATION criteria, and EXIT criteria. This approach is essentially a project control structure which has the property of (nearly) isolating problems until they are resolved. In practice, it represents one approach to continuous monitoring of product development.

For use in the type of process generation envisioned here, each quality technology process fragment definition would also include associated ETVX definitions.

3 SUMMARY

This section has presented a development architecture that incorporates an explicit link between software development process and software product quality. The link is provided by adjusting the development process to incorporate quality requirements in terms of quality factor levels and the associated developmental resources needed to correctly perform quality technology subprocesses. Validation of quality technology subprocesses is performed within the ETVX paradigm, and thus provides a high level of assurance that validation criteria will reflect specific quality requirements.

SECTION III

RESOLUTION of QUALITY FACTOR CONFLICTS

1 INTRODUCTION

1.1 The software development problem

The software development problem may be formulated as a problem in optimization where the **objective function** is:

> **[min** (cost, schedule), **max** (quality)],
> **subject to** Σ [requirements, resources, time and technology].

This is clearly an enormously difficult problem, one for which no optimal solution currently exists and for which none may ever be discovered. We have selected to study a subset of this problem, namely **max** (quality).

1.2 The software quality problem

The software quality problem can also be formulated as a problem in optimization:

> **max** [quality goals]
> **subject to** [technical feasibility, cost, schedule]

This also is a difficult problem which belongs to a class known as multiobjective or multiattribute optimization problems. These problems are characterized by the presence of multiple, conflicting goals accompanied by a large candidate solution space. The goals conflict because they are somehow interrelated. Another characteristic of this class is that no reallocation of resources can be made which will improve one objective without degrading at least one other objective as well. Due to this later characteristic, the solutions are known as non-dominated solutions. In the specific case of software quality goals, the non-dominance property may be relaxed in some cases but the spirit of the concept remains applicable.

In software development, there are several quality characteristics, or software quality factors, that inherently conflict. For example, efficiency and maintainability conflict if the objective is to improve code understandability, since efficiency frequently requires reliance on exceptional code. The same is true for expandability and reliability (increased risk to acquire more functionality), safety and availability (fail-soft/fail-safe requirements reduce the set of available system capabilities) and so on.

1.3 Conflict resolution

Determination of technical feasibility has been selected as a first research goal. Later, we plan to also incorporate cost and schedule feasibility considerations. We have started development of QUMAX, a set of tools to support planning and analysis procedures leading to QUality MAXimization. The development of QUMAX was recommended in our prior work as part of a software quality research architecture [2]. Our initial effort has produced a working prototype tool CORE (COnflict REsolution), which provides the following capabilities:

- Accepts as input an existing quantitative quality specification and checks the specification for technical feasibility. A quality specification is a set of quality factors together with a quantified level of desired quality for each factor in the set.

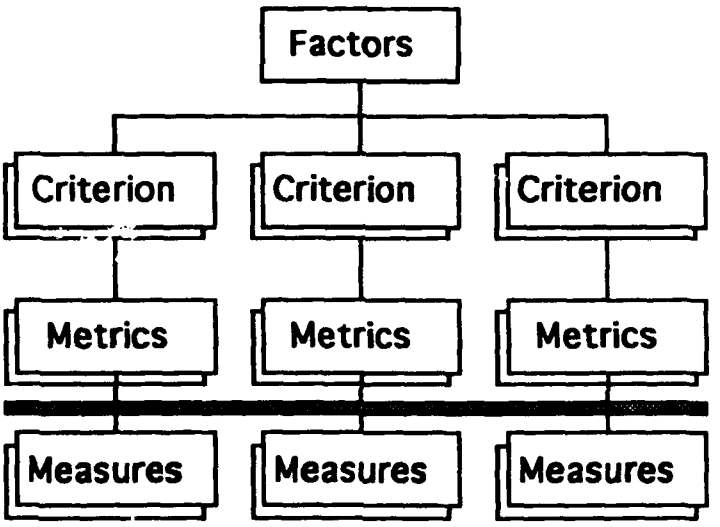
- If the initial quality specification is consistent (feasible), then no further action need be taken; otherwise, the user must adjust the quality specification. A consistent quality specification is one which does not violate the rules defining the nature and magnitude of factor interrelationships.

- The modified quality specification is checked for technical feasibility. If the specification is not feasible, then CORE will search for a feasible solution(s).

2 PRIOR WORK

2.1 Software Quality Framework

The program of software quality research funded by Rome Laboratory (formerly Rome Air Development Center) is based on a Software Quality Framework. The framework is the principal component of the Software Quality Methodology. This methodology defines procedures (a) for specifying prior to development a system's key quality factors and quantified factor goals, (b) for performing a technical feasibility and cost impact study, (c) for measuring the quality of intermediate software products, and (d) for measuring the actual achieved quality in the delivered software product. The Software Quality Methodology is defined in a three volume set of guidebooks entitled "Specification of Software Quality Attributes" [4]. The key concept of the framework is a three level hierarchical model of software quality.



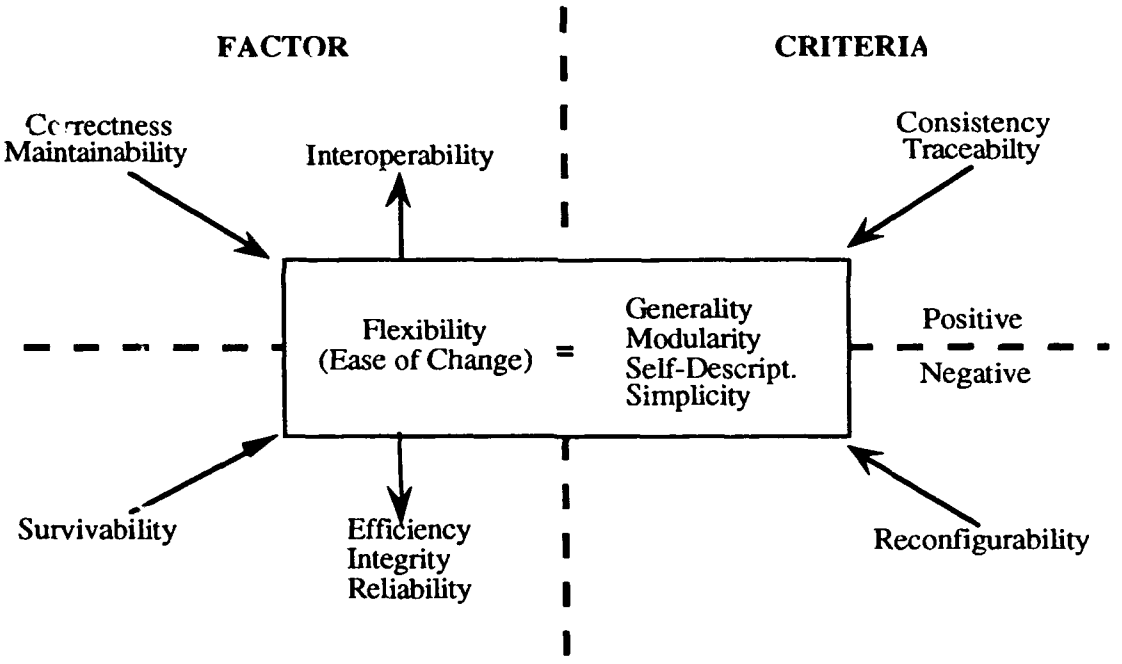
Software Quality Model. The level under the gray bar is usually not viewed as an additional level.

The top level is a set of thirteen customer-oriented *software quality factors*. The factor set is Efficiency, Integrity, Reliability, Survivability, Usability, Correctness, Maintainability, Verifiability, Expandability, Flexibility, Interoperability, Portability, and Reusability. The reader should keep in mind that the quality factors are not perfectly orthogonal; important relationships exist among the factor set.

In the current version of the Software Quality Framework, three levels of factor goals are identified: excellent, good and average. For quantitative goal setting, excellent is mapped into .90 to 1.00, good into .80 to .89, and average into .70 to .79. The role of these factor goals in the overall methodology is considerable. For example, at the end of the development, a score is calculated, in the range of 0.0 to 1.0, which represents how much of a certain quality factor is actually present in the final product. That final score is then compared to the target goal for that factor to determine if the contractor has meet requirements.

The second level is a larger set (twenty-nine) of defining attributes for the software quality factors. These are termed *software quality criteria* and reflect technical (developer) considerations of good software engineering and development practice. For example, the RELIABILITY software quality factor is defined in terms of three software quality criteria: accuracy, anomaly management and simplicity. Some criteria are used in the definition of more than one factor. The reader should keep in mind that the quality criteria are not perfectly orthogonal; important relationships exist among the criteria set.

The complexity of quality factor/quality criteria interrelationships is apparent in figure shown below which is a representation of interrelationships for the FLEXIBILITY quality factor. This form of representation was developed by others in the course of conducting a prior assessment of the framework [13].



Factor/Criteria Interrelationships for FLEXIBILITY

Figure 2 depicts seven types of relationships:

1. Positive factors impacting FLEXIBILITY (Correctness, Maintainability)
2. Negative factors impacting FLEXIBILITY (Survivability)
3. Positive criteria impacting FLEXIBILITY (Consistency, Traceability)
4. Negative criteria impacting FLEXIBILITY (Reconfigurability)

5. FLEXIBILITY positively impacting other factors (Interoperability)

6. FLEXIBILITY negatively impacting other factors (Efficiency, Integrity, Reliability, Survivability)

7. Criteria which define FLEXIBILITY (Generality, Modularity, Self-Descript, Simplicity)

If one were to produce a set of connected graphs for the complete set of quality factors and quality criteria were shown, it would be clear that summation over all criteria, factor and criteria/factor relationships produces a highly complex web of software quality interrelationships.

The explicit recognition of technical interrelationships among factors and between factors and criteria in the Software Quality Framework is unique among software quality models. It provides the basis for a systems engineering approach for specifying software quality requirements.

These technical relationships are either positive (beneficial) or negative (adverse). At the factor level, if factor X positively impacts factor Y, then the presence of factor X will increase the likelihood of achieving the desired quality goal for factor Y. If the indicated relationship is negative, then the presence of Factor X will increase the difficulty of achieving the desired quality goal for Factor Y. For example, their is a defined negative relationship between the factors Efficiency and Maintainability. That means that it will be difficult to achieve the target goals if both factors are assigned a level of Excellent (High). The rationale is that high hardware efficiency is often obtained by use of programming practices which impair future maintainability. The Software Quality Framework also defines the relative strength of the interrelationships in terms of degree of impact. Not all pairs of factors are interrelated. If they were, a nearly optimal tradeoff resolution of factor conflicts would be virtually impossible. In many cases, the impact is one to many, so that factor Y above could also represent a set of factors. Similar reasoning applies to interrelationships between factors and criteria.

2.2 Assistant for Specifying the Quality of Software

The Rome Laboratory software quality research program also developed a tool to provide assistance during the process of developing a quality goal specification. The Assistant for the Specification of Software (ASQS) is an expert system whose principal objective is to facilitate the transition of the Software Quality Methodology into acquisition management practice. The underlying strategy to accomplish this objective was to provide facilities which partially automated the software quality specification process as outlined in Guidebook Volume II. The ASQS was developed by Dynamics Research Corporation during the period 1985-1990. The current version of ASQS should be considered a proof-of-concept demonstration. Interested readers should consult the ASQS Operational Concept Document [3] and the ASQS User's Manual [14].

The rationale which lead to the development of the ASQS was that system acquisition managers are typically unfamiliar with software quality concepts and technology. So, they would need assistance in translating their knowledge of software-intensive system characteristics and requirements into a software quality specification. In addition, the technical and feasibility and cost impact analyses, which are an integral component of the Software Quality Methodology, would likely be sufficiently complex that computer support would be required for all but the simplest systems.

The ASQS implementation of the technical feasibility and cost impact analyses was incomplete. The development of CORE is an attempt to begin to complete the ASQS proof-of-concept demonstration. CORE is presently a standalone system and their are no plans to integrate CORE into the existing ASQS.

3 OVERVIEW of CORE

CORE (COnflict REsolution) is a decision support system for identifying and resolving critical software quality factor conflicts. CORE implements an interpretation of the Software Quality Methodology described in Volume II of [4]. Knowledge about factor and criteria interrelationships contained in Volume II have been encoded into CORE as sets of Prolog facts. CORE was originally developed in C-Prolog, an interpretive Prolog development language. Subsequently, CORE was ported to MacProlog, a Prolog development system which generates compiled code. A Macintosh interface was also developed for CORE using the MacProlog development environment.

This section, which assumes a basic familiarity with the contents of Volume II, consists of a description of CORE, a scenario which follows the Airborne Radar System example contained in Volume II, and the corresponding output report. The example system is described in Table 4.1.1-1, Characteristics and Functions for Example System, page 4-10. All page and table references are to Volume II of *Specification of Software Quality Attributes.*

3.1 Input

A user may select any of the initial factor goal sets listed in Table 4.1.2-7, Software Quality Factor Identification Form-Initial Goals, page 4-24, or may modify any of the initial goal sets. Similarly, the user may use any of the default factor formulas listed in Table 4.2-1, Criteria Weighting Formula Form-Initial Weighting, page 4-64, or the user may modify the criteria weights of the default formulas. Finally, the user may initialize any subset of criteria to values which the user considers minimally acceptable.

3.2 Processing

After the input phase has been completed, CORE:

(1). computes the values of criteria which have not been initialized by the user and lists all formulas and constituent criteria values for the user's inspection.

(2). computes the adverse effects of factors on each other,

(3). computes the beneficial effects of factors on each other,

(4). lists all factors together with their actual and maximum permissible (achievable) values.

(5). For any factor F whose actual value exceeds its achievable maximum, the system:

(a) lists all factors which negatively impact F,

(b) determines the extent of impact,

(c) lists the factor constituents (i.e. criteria) responsible for the negative impact,

(6). If no critical factor conflicts are uncovered, the session terminates. Otherwise, CORE asks the user to optionally enter minimum acceptable values for any of the individual

factors. CORE also takes a snapshot of the data base as it exists just after factor minima have been assigned.

(7). CORE then initiates a search for solutions to factor conflicts. The general strategy is to selectively reduce the values of factors responsible for conflicts.

(8). At this point, non-default minimum values (i.e., those factor minima set by the user), become the actual values of those factors. (The rationale for this is discussed below; see Factor Minima). The initial task is to determine whether the user has established minima low enough to resolve the conflict. If so, there is no point in continuing the search for solutions. Otherwise the following algorithm is implemented:

1. Identify the set of factors which are involved in the conflict. This is the conflict set. In practice, this is the set consisting of the factor whose permissible maximum has been exceeded together with all factors (and implicitly all criteria) which have an adverse effect on that factor.

While no solution has been found or the user chooses to search for additional solutions:

2. For each member of the conflict set with a value of Excellent or Good,

   a. reduce the factor value one level (to Good or Average),
   b. recalculate all factor values, criteria values, and interrelationships,
   c. determine whether the reduction of that factor was sufficient to resolve the conflict.
   d. If yes, report details of the resolution and optionally continue the search.
   e. If no, continue the search.

3. For each factor combination X,Y from the conflict set such that both X and Y have values greater than Average, and neither X nor Y is by itself sufficient to resolve the conflict (see step 2),

   a. reduce the values of X and Y one level,
   b. recalculate all factor values, criteria values, and interrelationships,
   c. determine whether the reduction of X and Y was sufficient to resolve the conflict.
   d. If yes, report details of the resolution and either continue or terminate the search at the discretion of the user.
   e. If no, continue the search.

4. Repeat step 3 for each combination X,Y,Z from the conflict set such that X, Y and Z have above average values and no subset of X, Y and Z is sufficient to resolve the conflict (see steps 2 and 3).

5. If the adversely affected factor is efficiency, continue the search using combinations of four and then five factors. (No other conflict set is large enough to warrant this step.)

6. After the search phase has been completed, a report is generated for the user displaying a snapshot of the data base at the time the conflict was discovered and describing solutions discovered in the course of the search. Each solution includes explicit directions about which factors to reduce and how to reduce them (from Excellent to Good or from Good to Average.) In addition, each solution displays the factor values and maxima that would result if that particular solution were adopted and if all factor interdependencies were recalculated. A second report format provides a more detailed snapshot, including formulas and the values of their component criteria as well as factors, values and maxima.

3.3 Execution Time:

Example CORE timing results were obtained on a 16 MB Mac IIfx and CORE implemented in MacProlog. The search times are shown on program screens which display candidate solutions to the example problem; see section 4.

3.4 Criteria

Criteria may be either shared by more than one factor or unique to a single factor. Default values for shared criteria are 0.85 ('good' or 'moderately important'). Values for unique criteria are computed from their factor values and shared criteria values. For example:

The formula for reliability is 0.4*ac + 0.3*am + 0.3*si. Suppose that reliability has been assigned a value of 0.90, and that its two shared criteria, am and si, have the default value 0.85. The value of the unique criterion, ac, can then be determined to be 0.97, since 0.90 = 0.4*0.97 + 0.3*0.85 +0.3*0.85. Here is a general algorithm.

(1). Sum the weighted shared criteria.

(2). Subtract the result from the formula value.

(3). Subtract the weights of the shared criteria from 1.0.

(4). Determine the value of remaining criteria by dividing the results of steps 2 & 3, and dividing that result by the number of remaining criteria.

It could (and often does) happen that the result of step 4 is greater than 1.0. This means that there is no way of satisfying the formula unless (a) the shared criteria values are raised and the unique criteria values lowered or (b) the factor value is lowered, or (c) the weights of the shared criteria are lowered, those of the unique criteria raised, and the values of the unique criteria lowered. (c) is the preferred course of action, since it is least likely to override the preferences of the user. (c) is the method used in the current CORE implementation. The system repeatedly reduces the shared weights by 0.05 until acceptable results are achieved. Here is an example. Suppose that the desired value of the factor reliability is 0.95, and that the values of its two shared criteria, anomaly management and simplicity, are the default values, 0.85 and 0.85. In this case its third (non-shared) criteria will have an initial computed value of 1.225, since:

Rel(0.95) = [ac](0.4)(1.225) + [am](0.3)(0.85) + [si](0.3)(0.85).

By repeatedly lowering the weights of am and si by 0.05, each time increasing the weight of ac by 0.10, the following result is achieved:

Rel(0.95) = [ac](0.7)(0.99) + [am](0.15)(0.85) + [si](0.15)(0.85)

The advantage of this solution is that the factor value as well as the shared criterion values remain unchanged. The disadvantage, of course, is that factor formula is modified, and perhaps significantly so. This explains why default values for shared criteria are so high (0.85). Lower values would result in formulas whose weights would be very different from those specified in Volume II.

Before leaving this topic, one other situation is noteworthy. It could happen that all criteria for a given formula have a value greater than 0 and yet the sum of the weighted criteria is not equal to the value currently assigned to the corresponding factor. The system resolves the

situation as follows. If the factor value is greater than the weighted criteria sum, the factor value is reduced. But if the factor value is less than the weighted criteria sum, the values of the component criteria are reduced.

The advantage of this method of resolution is that it is consistent with the way that such inequalities typically arise. The system reduces factor values in an attempt to find solutions to critical factor conflicts. When this happens, a proportional reduction in the criteria which comprise those factors is clearly in order. But some of the lowered criteria may well be shared by other factors, so that as a side effect, the weighted criteria sums of the corresponding factor formulas will fall below the factor values. To avoid infinite regress, these factor values must be reduced in such cases.

3.5 Quantifying Factor Interrelationships

Factor interrelationships implemented in this system are those to be gleaned from Table 4.1.3-1, Effects of Criteria on Software Quality Factors, page 4-26, together with Table 4.1.3-2, Positive Factor Interrelationships, pages 4-28 to 4-31, and Table 4.1.3-3, Negative Factor Interrelationships, pages 4-32 to 4-33. Although adverse and beneficial relationships cannot be quantified with great precision, CORE does succeed in closely simulating the two major conflicts discussed in the text. Both examples assume the Surveillance and Identification goal set.

Table 4.1.3-4, Factor Interrelationship Calculations, page 4-34, indicates that the combined negative impact of the adaptation factors on efficiency is -10. However, since all adaptation factors have a value of 'good' (0.80 - 0.89), the conflict is "not critical" (section 4.1.3.4, page 4-35). Table 4.1.3-4 also indicates that integrity, reliability, usability, maintainability, and verifiability also have a combined negative impact on efficiency of -10. But in this case, the conflict "is critical", since the values for these factors is excellent (0.9 - 1.0). "This situation...requires action because achieving the initial set of goals is not possible" (section 4.1.3.4, page 4-35).

It follows that the effect of one factor on another depends both on the degree of impact (ranging from 1 to 3) and the rank of the affecting factor (ranging from 0.7 (Average) to 1.0 (Excellent)). In light of this, the following scheme was adopted.

Assume that F1 negatively affects F2.

(1) If the value of F1 is in the excellent range, F1 reduces the maximum value of F2 by 0.015 * the degree of impact (1 - 3).

(2) If the value of F1 is in the good range, F1 reduces the maximum value of F2 by 0.0075 * the degree of impact (1 - 3).

(3) If the value of F1 is in the average range, F1 reduces the maximum value of F2 by 0.005 * the degree of impact (1 -3).

Applying these rules, the adaptation factors with a combined impact of -10, reduce the potential maximum of efficiency to 0.925 (i.e. 1.0 - 10 * 0.0075). Since it is still possible to achieve an excellent rating for efficiency, the conflict is not critical. On the other hand, the excellent factors conflicting with efficiency, again with a combined impact of -10, reduce the potential maximum of efficiency to 0.85 (i.e. 1.0 - 10 * 0.015). Hence this conflict is critical, since it is not possible to achieve excellence among all factors involved in the conflict.

A troublesome question remains. Are the 'conflicts' under discussion independent conflicts which can be considered in isolation as is, I think, suggested by the authors of Volume II ; or are they instead two aspects of a single conflict involving efficiency on the one hand and all factors which adversely affect efficiency on the other? We believe that the first view is fraught with difficulty. For this reason, CORE has been designed to reflect the second view.

At the time this particular conflict is discovered, CORE displays the following screen of information.

Efficiency = 0.90 which exceeds the permissible maximum 0.78 . The chart below lists factors which reduce maximum efficiency from 1.00 to 0.78 together with their values and the extent to which each contributes to the reduction.

Ok

Factor	Value	Percent
Integrity	0.90	3.00
Reliability	0.90	4.50
Usability	0.90	1.50
Maintainability	0.90	3.00
Verifiability	0.90	3.00
Expandability	0.80	3.00
Flexibility	0.90	1.50
Interoperability	0.80	3.00

The adaptation factors, expandability, flexibility and interoperability, reduce the maximum permissible value of efficiency by 7.5%. Integrity, reliability, usability, maintainability and verifiability reduce the maximum permissible value of efficiency by 15%. Thus the combined impact of all factors which negatively affect efficiency is to reduce the maximum permissible value of efficiency by 22.5% (rounded to 22% above). This implies that given the current factor values, the maximum permissible value of efficiency is 0.78 (on a scale of 0.0 to 1.0).

Note that the actual value of flexibility is now 0.9, whereas it was initialized at 0.8. That is because there exists a conflict at another level in the original goal set. The system recognized this and made the appropriate adjustment. The problem here is that all constituent criteria of flexibility are shared. To achieve the 0.9 factor values required by the Surveillance and Identification goal set, the system had to assign very high values to the criteria generality, modularity, self-descriptiveness and simplicity, among others. As a side effect, the value of flexibility was sharply increased, since these are just the criteria which collectively define flexibility. One strength of CORE is that it enforces constraints imposed by shared criteria

values on factor values, detecting and correcting violations of those constraints in a user's choice of goal sets.

3.6 Factor Minima

After a critical factor conflict is uncovered, and just prior to initiating the search for conflict resolutions, the user is allowed to set minimum values for factors. When a factor is assigned a minimum value, its actual value is made equal to that minimum. In addition, the values of all criteria within the formula for that factor are adjusted accordingly.

Setting a minimum value for a factor is a two edged sword. On the one hand, it guarantees that the value will not fall below that minimum. On the other hand, it has the side effect of making the actual value equal to the minimum value. For example, if the value for efficiency is 0.9 and the user at this point indicates that the minimum acceptable value is 0.85, the system will reduce the value of efficiency to 0.85. This may appear extreme, but there is a good reason for it. If the user wishes to resolve a factor conflict and has indicated a willingness to accept a reduction in efficiency to 0.85, then why not facilitate the search for solutions by putting that reduction into effect immediately. The user may accept the minimum only grudgingly, much preferring a higher value for that factor. But there is a simple remedy at hand. Simply set the minimum to that higher value and search for a resolution to the conflict given that more stringent requirement.

Factor minima can be used to control and constrain searches. For example, suppose that one wished to determine whether a particular conflict could be resolved by reducing factors X and Y and leaving all others unchanged. Further suppose that X's value was Excellent and Y's value Good. To test the hypothesis, one would

1. Choose "Set minima to Current Factor Values" from the factor minimum menu, setting all minimum values to their actual factor values.
2. Then reduce X's minimum to 0.88 and Y's to 0.78.
3. Choose "Exit Menu"

Only one solution to the conflict is now possible and that is the one hypothesized.

A more flexible method for constraining searches permits one to determine whether a solution can be found among some particular subset of factors, say X, Y and Z, without lowering the values of factors outside that subset.

1. With the exception of X, Y and Z, set each factor minimum to its current value.
2. Choose "Exit Menu"

Now only solutions involving reductions in the three factors of interest will be considered.

4. Sample Session with CORE

The following sequence of screens illustrates a CORE session for the Surveillance and Identification function of the Airborne Radar System example. The decision to lower the goal of the efficiency quality factor is simulated in the CORE session. A detailed report of the session results follows the sample session.

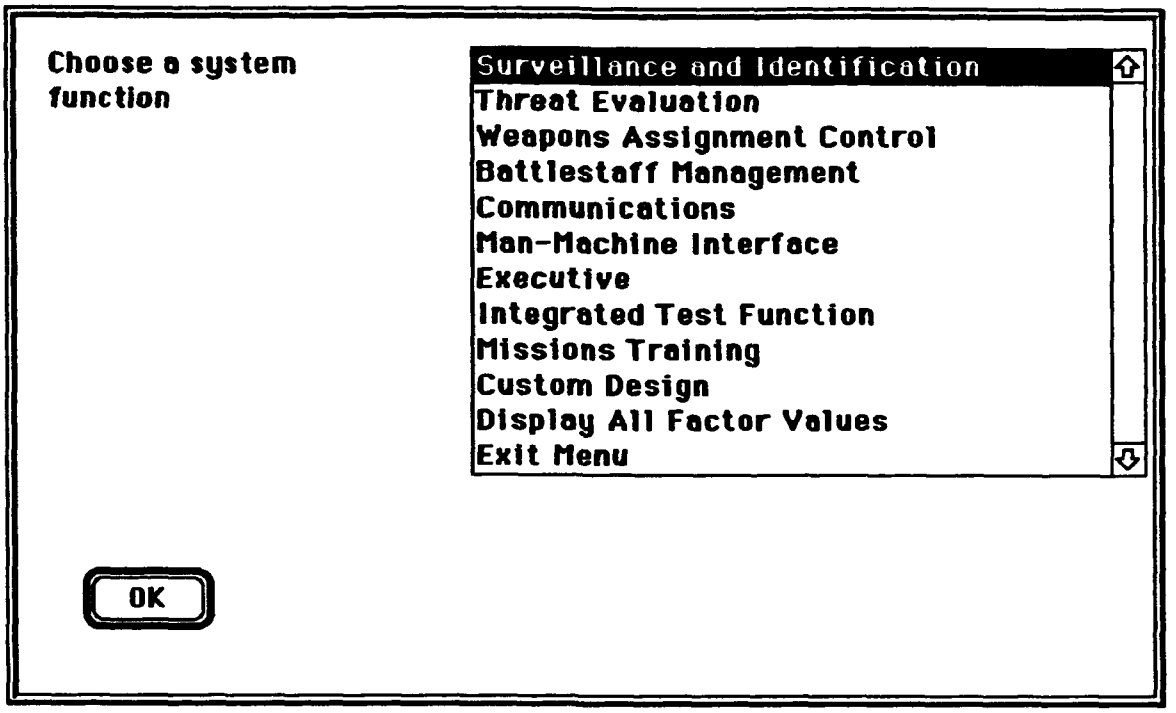


Figure 1. Select a system function.

This screen is used to select a factor goal set. The goal sets shown are taken from Table 4.1.1-1, page 4-10. This example scenario is based on the Surveillance and Identification function.

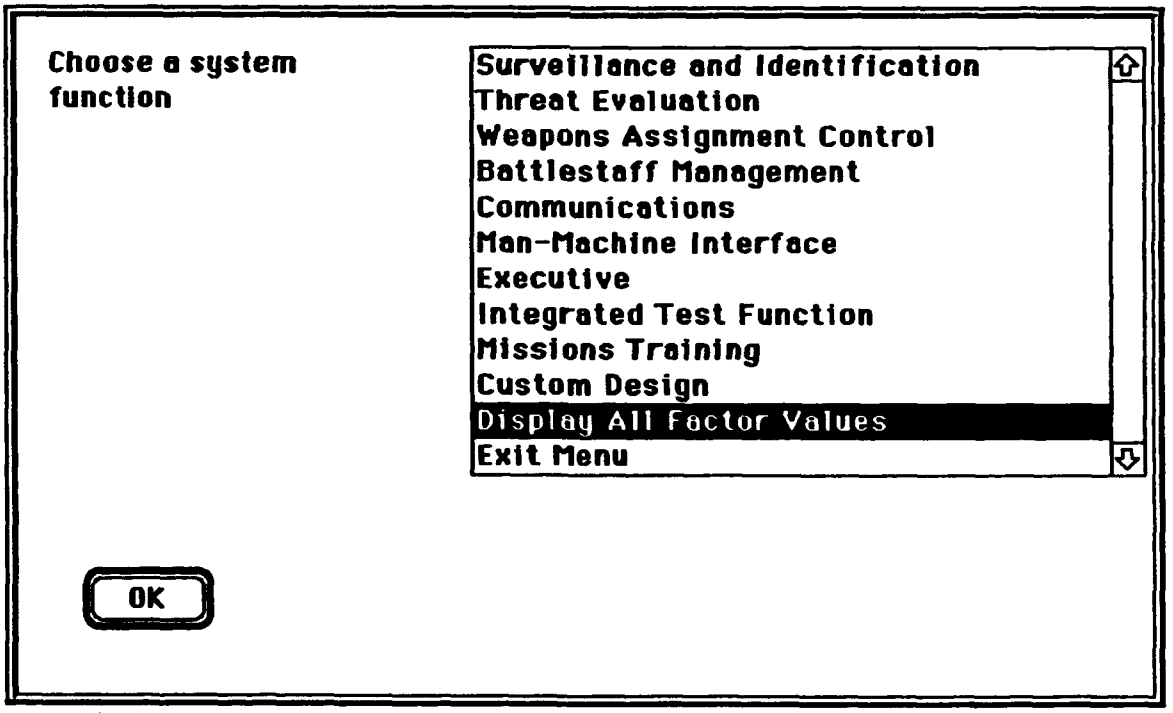


Figure 2A. Display all factor values.

This menu item will result in a display of the factor value set for Surveillance and Identification.

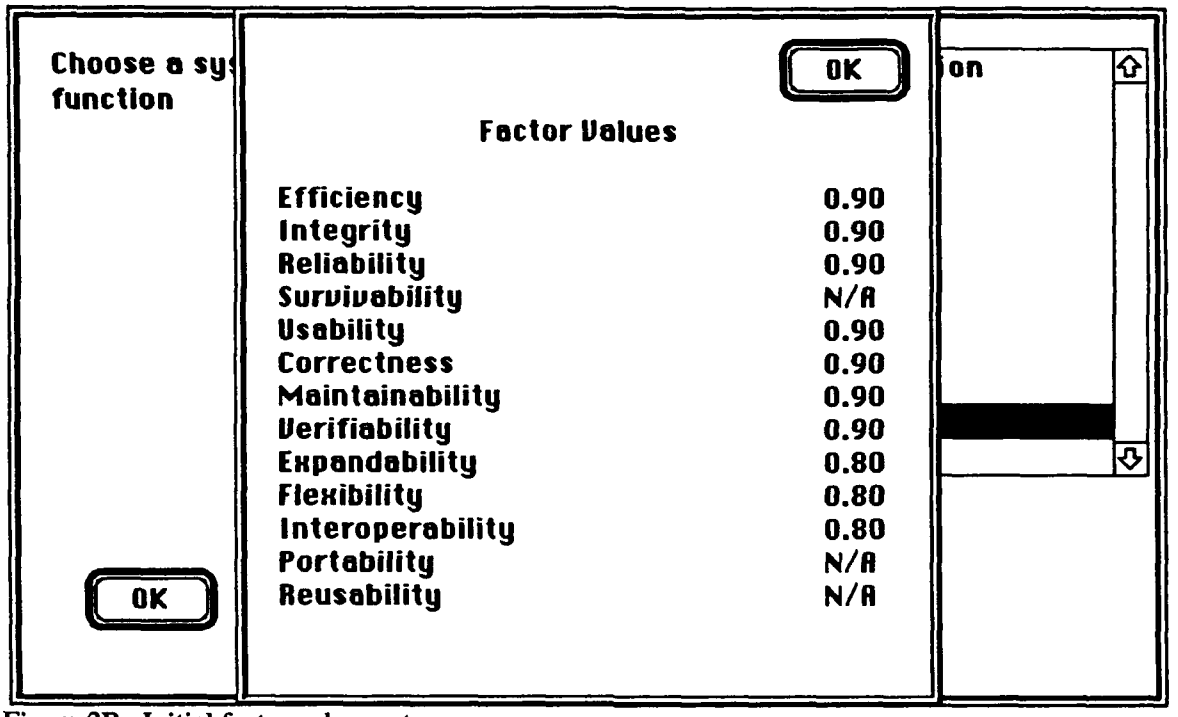


Figure 2B. Initial factor values set.

CORE assigns a value of 0.90 to an initial quality goal of excellent and 0.80 to an initial quality goal of good. The initial quality goals shown are taken from Table 4.1.2-7, page 4-24. N/A is not important or not applicable.

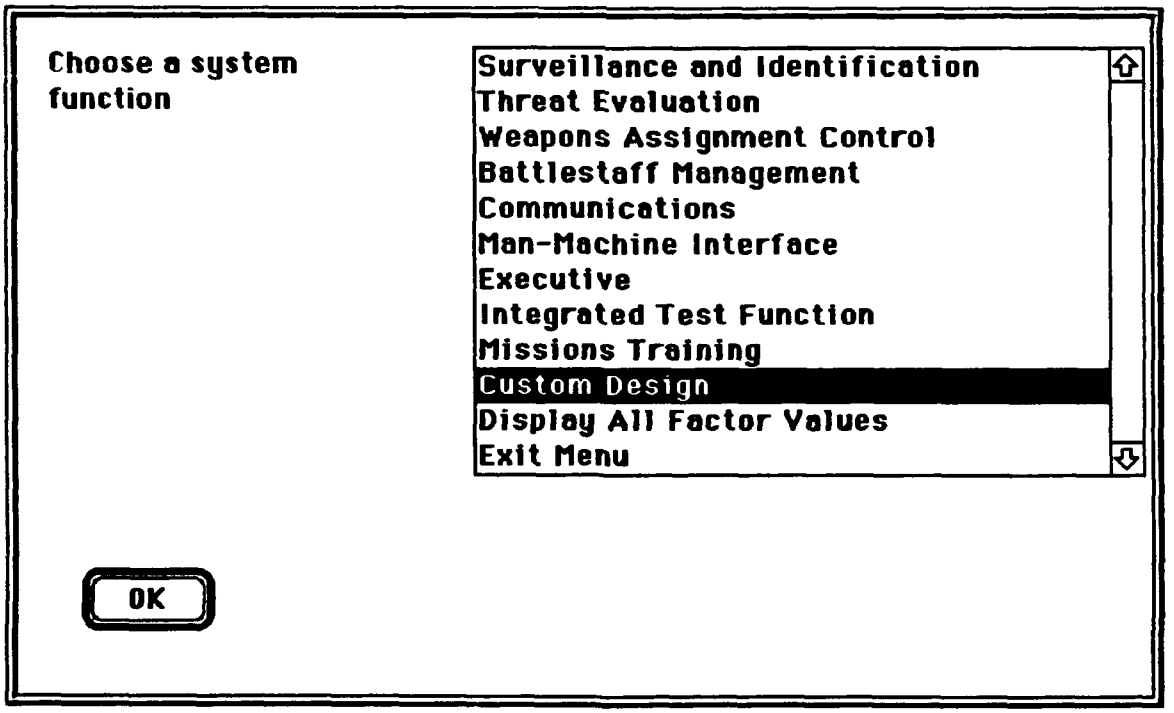


Figure 3A. Custom Design.

This menu item provides a capability to modify initial baseline factor goals for any of the selected factor sets.

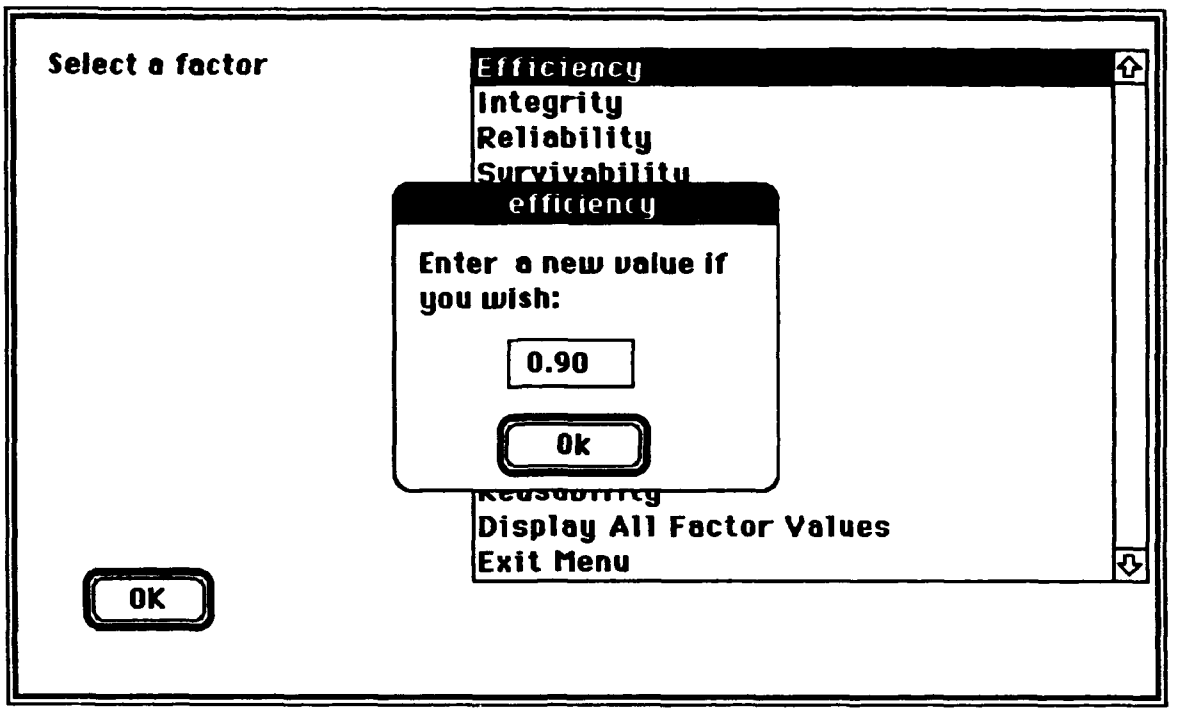


Figure 3B. Change factor goal values.

Sub-menu providing the capability to enter a new quality goal for the efficiency quality factor. The baseline value for efficiency is displayed, and in this example, is not changed.

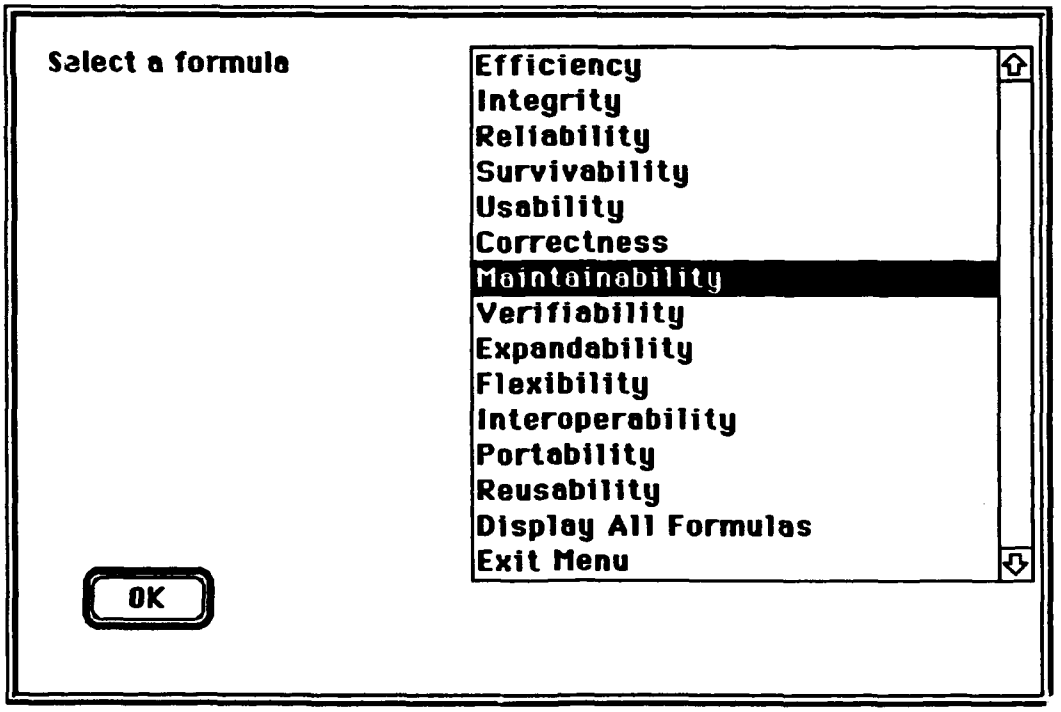


Figure 4A. Select a formula.

This screen provides a capability to select a factor in order to modify criteria weights in the factor formula. Criteria cannot be added or deleted.

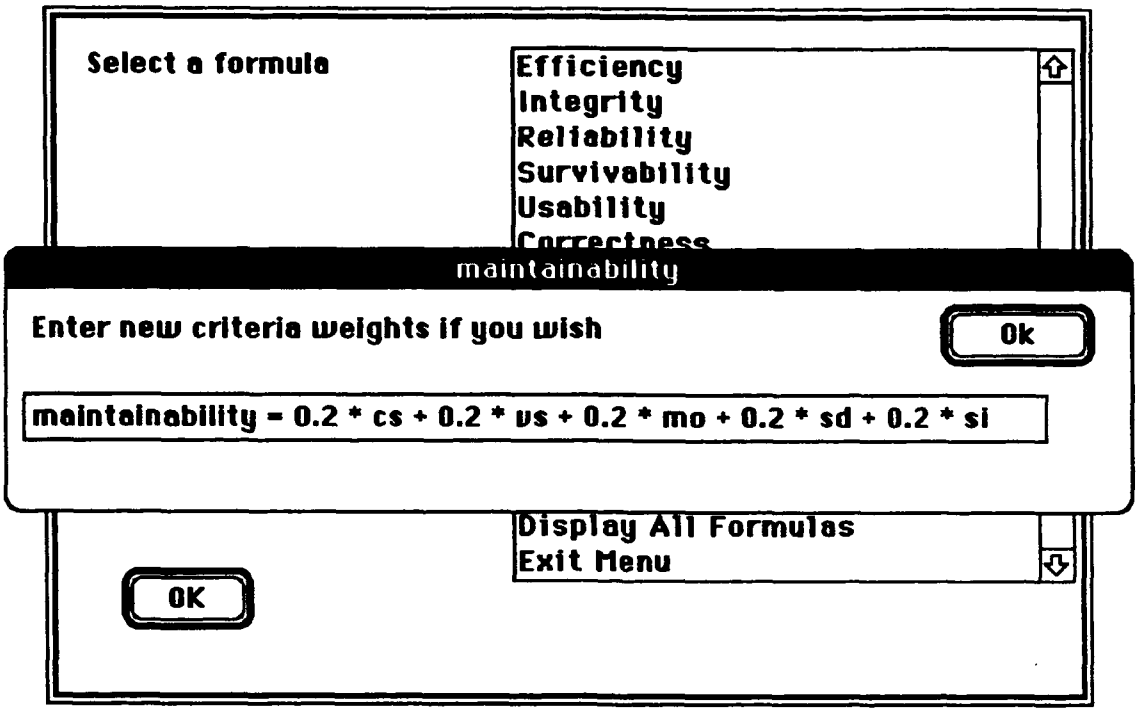


Figure 4B. Modify criteria weights..

Sub-menu providing a capability to modify criteria weights. In this example, no criteria weights are changed.

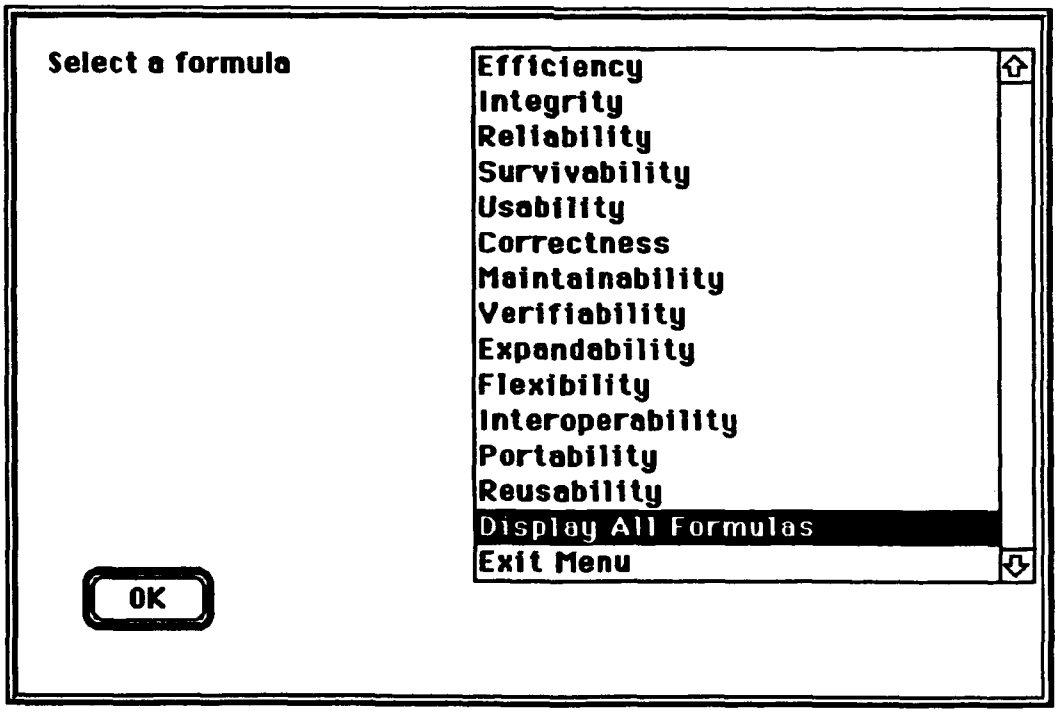


Figure 5A. Display all factor formulas.

This menu item provides a capability to select all factor formulas for review.

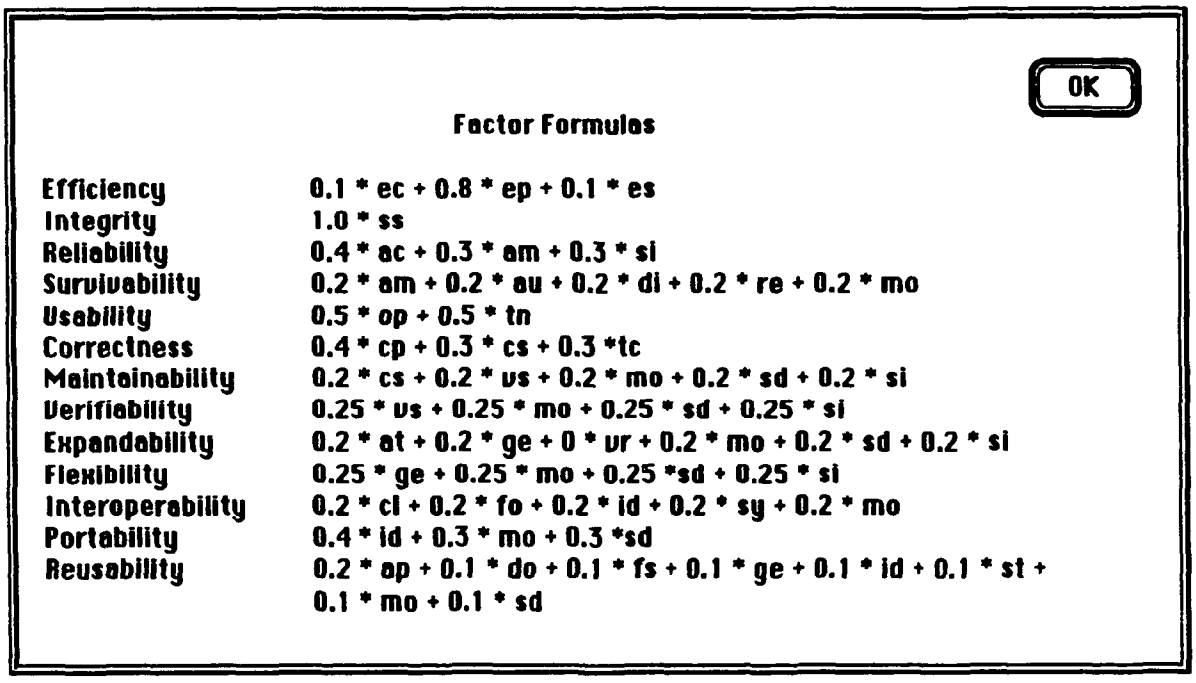


Figure 5B. Factor formulas.

This screen provides a capability to review all factor formulas.

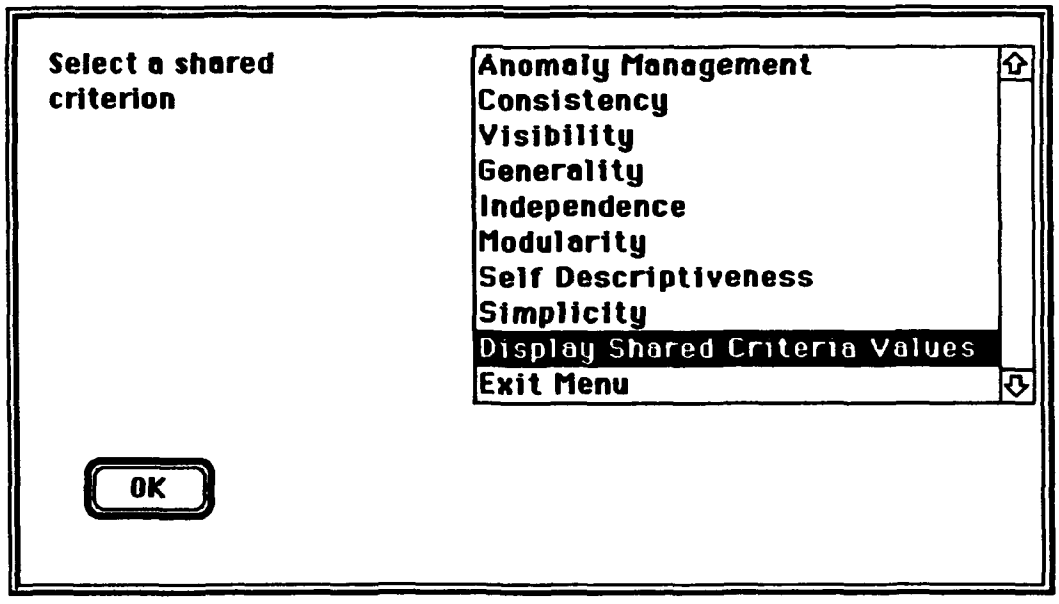


Figure 6A. Select a shared criterion.

This screen provides a capability to change shared criterion values and/or to display them. In this example, no shared criteria are changed.

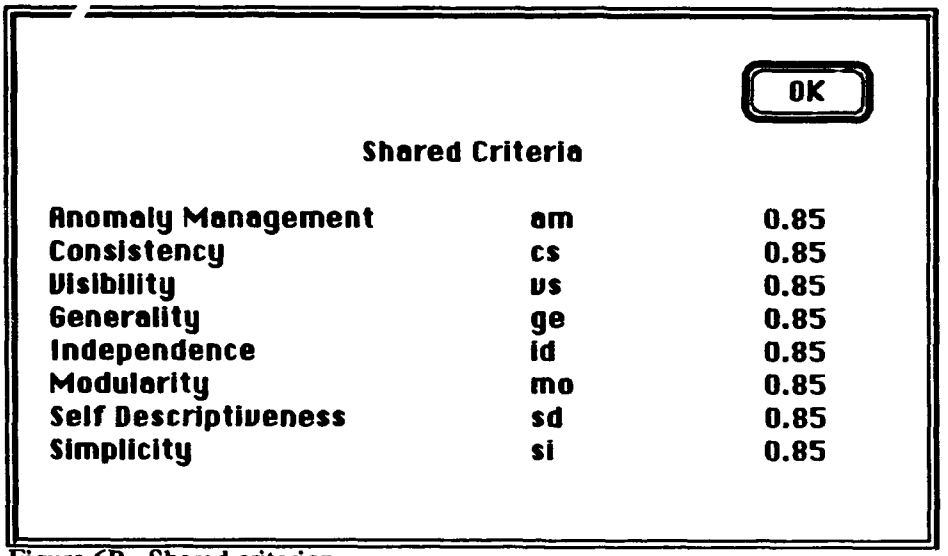


Figure 6B. Shared criterion.

This screen provides a capability to review all shared criteria values.

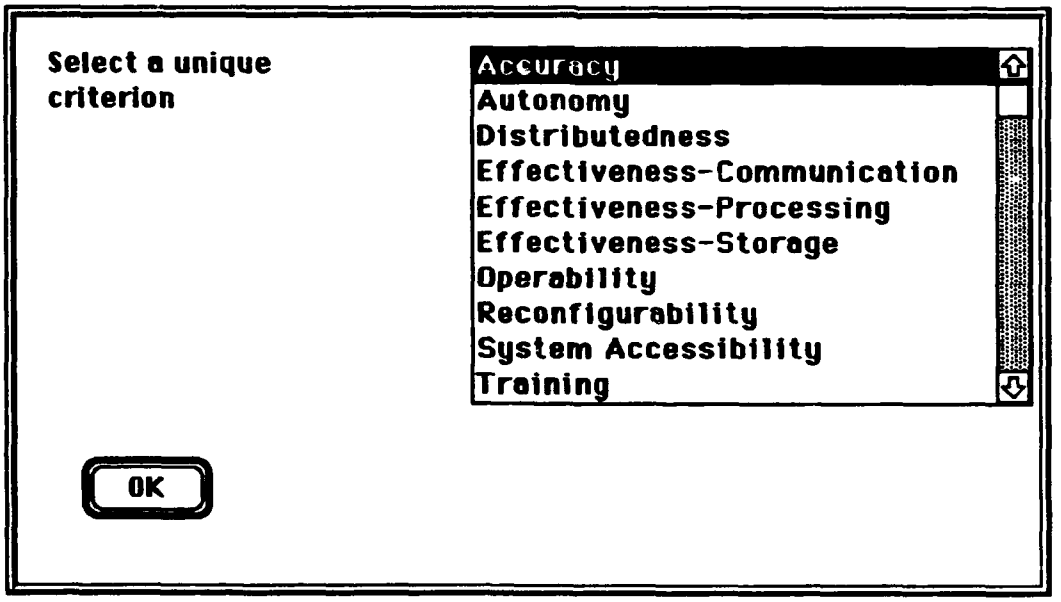


Figure 7A. Select a unique criterion.

This screen provides a capability to assign minimum acceptable values to specific criteria. The remaining criteria are accessed by scrolling.

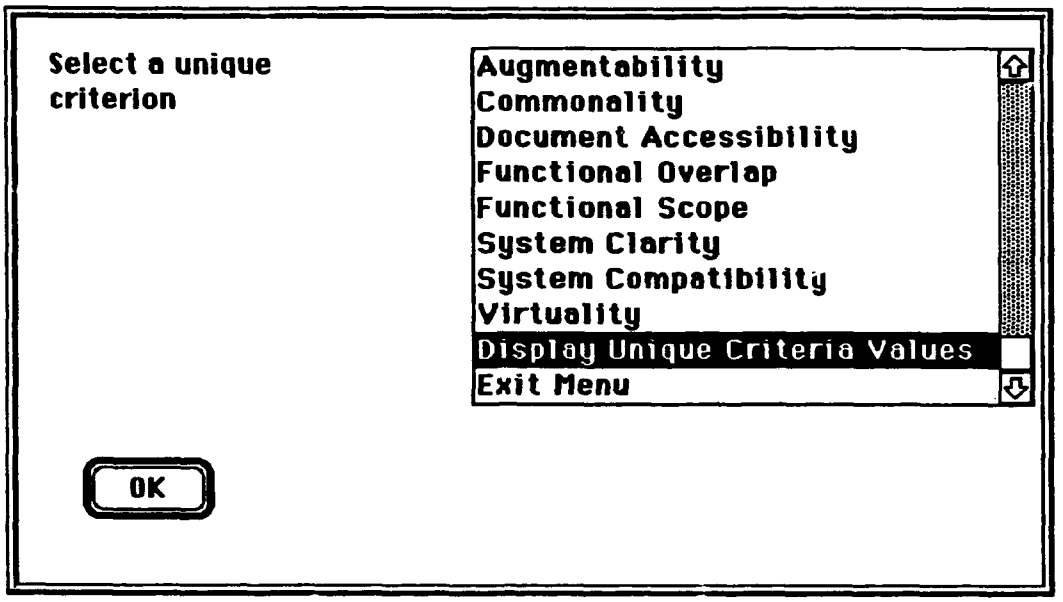


Figure 7B. Scrolled from 7A..

In this case, the capability to display all unique criteria values is selected.

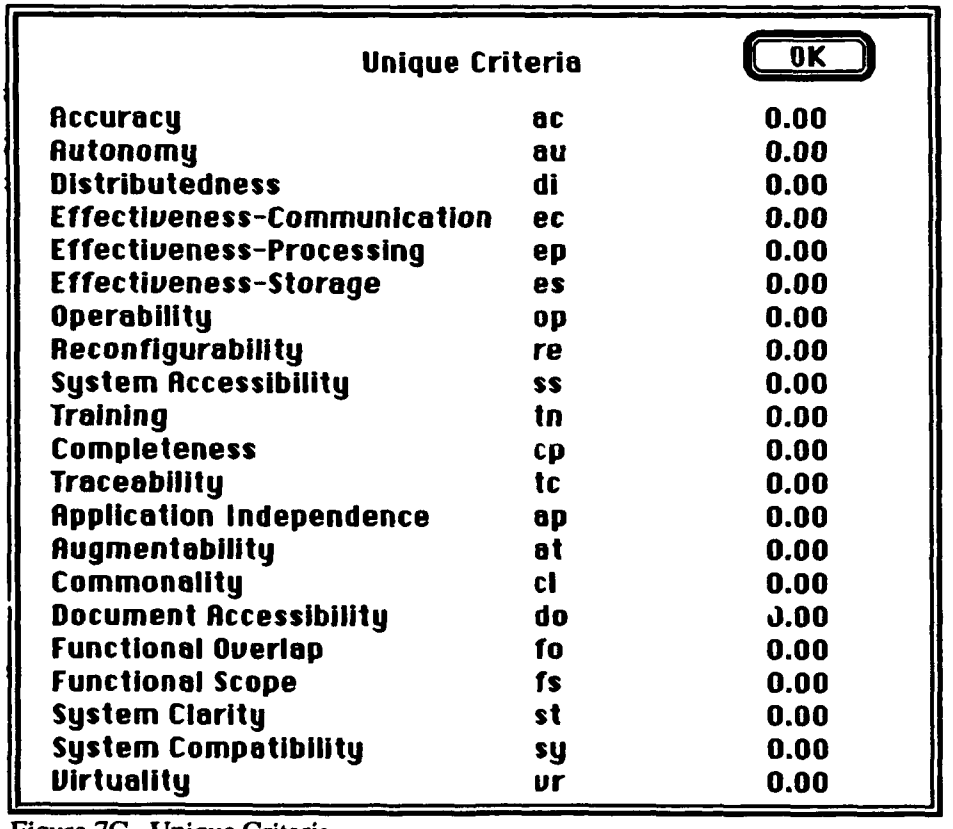


Figure 7C. Unique Criteria.

This screen provides a capability to review all unique criteria values. In this example, all values are equal to zero because no minimum values have been assigned and CORE has not yet initiated any calculations.

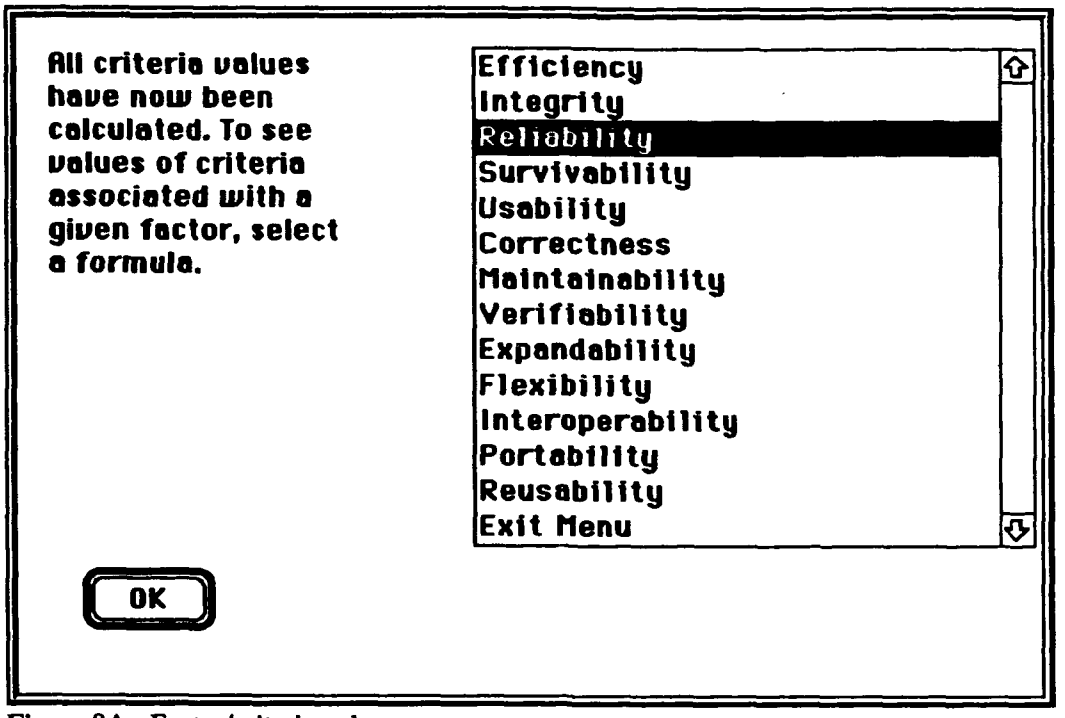


Figure 8A. Factor/criteria values.

At this point, CORE has calculated all criteria, factor and formula values. In addition, positive and negative interrelationships have been taken into account. This screen provides a capability to select for review the resultant factor values and their associated criteria values.

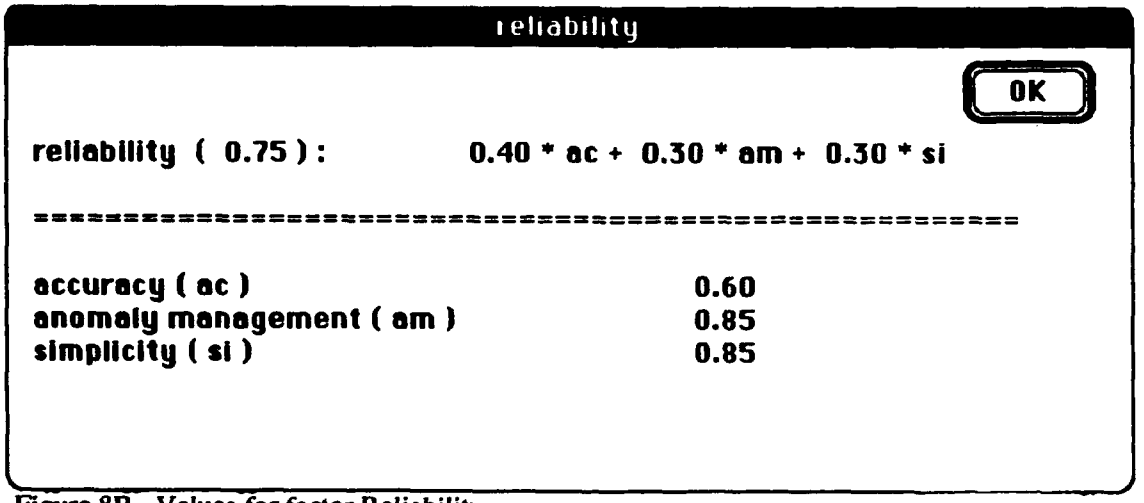


Figure 8B. Values for factor Reliability

This screen provides a capability to review the calculated values for factors and their associated criteria values.

Ok

Factor Values & Permissible Maxima

Factor	Value	Maximum
Efficiency	0.90	0.78
Integrity	0.90	0.97
Reliability	0.90	0.98
Survivability	N/A	0.98
Usability	0.90	1.00
Correctness	0.90	1.00
Maintainabilit	0.90	1.00
Verifiability	0.90	1.00
Expandability	0.80	1.00
Flexibility	0.90	1.00
Interoperability	0.80	1.00
Portability	N/A	0.97
Reusability	N/A	1.00

Figure 9. Desired vs. achievable factor values.

CORE automatically generates this screen which shows the target factor goals (Value column) and the maximum achievable goals given the factor/criteria interrelationships (Maximum column). In this case, the target goal for efficiency (.90) exceeds the maximum achievable goal (0.78). Thus, this is an inconsistent goal set.. In order to specify a consistent goal set, a reduction in target goals must be assigned to one or more factors.

Efficiency = 0.90 which exceeds the permissible maximum 0.78 . The chart below lists factors which reduce maximum efficiency from 1.00 to 0.78 together with their values and the extent to which each contributes to the reduction.

Ok

Factor	Value	Percent
Integrity	0.90	3.00
Reliability	0.90	4.50
Usability	0.90	1.50
Maintainability	0.90	3.00
Verifiability	0.90	3.00
Expandability	0.80	3.00
Flexibility	0.90	1.50
Interoperability	0.80	3.00

Figure 10. Conflicting factors.

CORE automatically generates this screen which shows the negative impact on the efficiency factor caused by other interrelated factors. The percent column sums to 0.22; 1.00-0.22 = 0.78, the calculated maximum achievable value for efficiency.

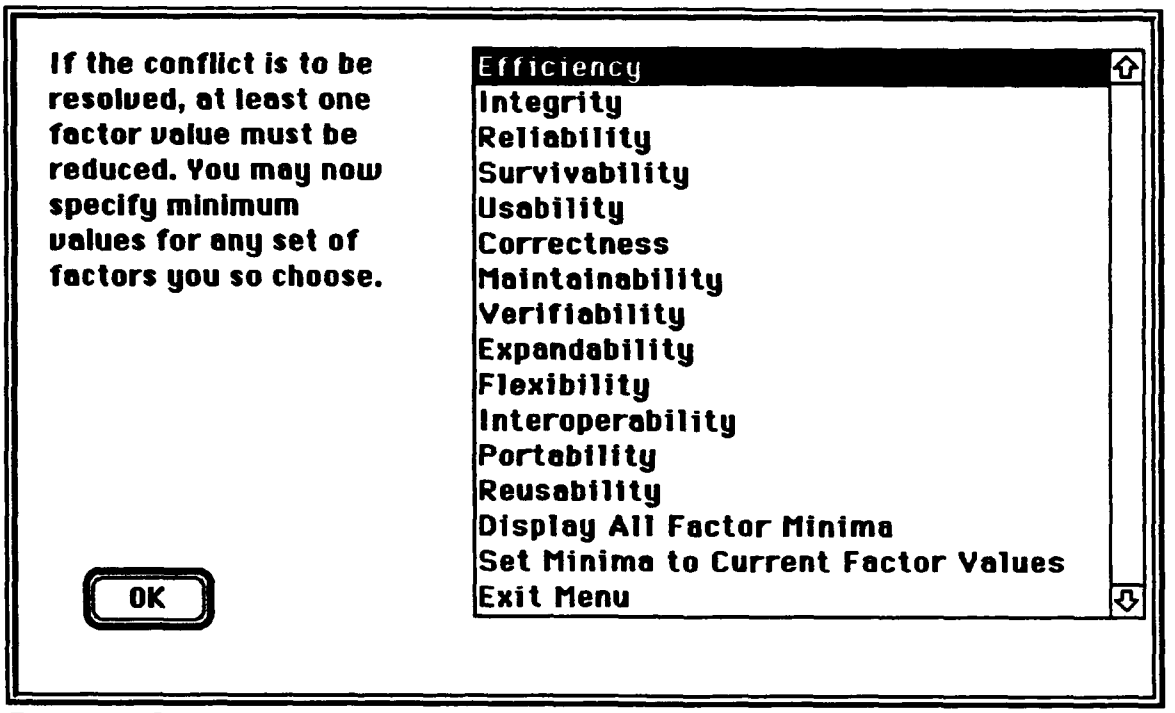


Figure 11. Conflict resolution choice.

This screen provides the capability to select which factor(s) goals the user is willing to modify in order to create an achievable factor goal set.

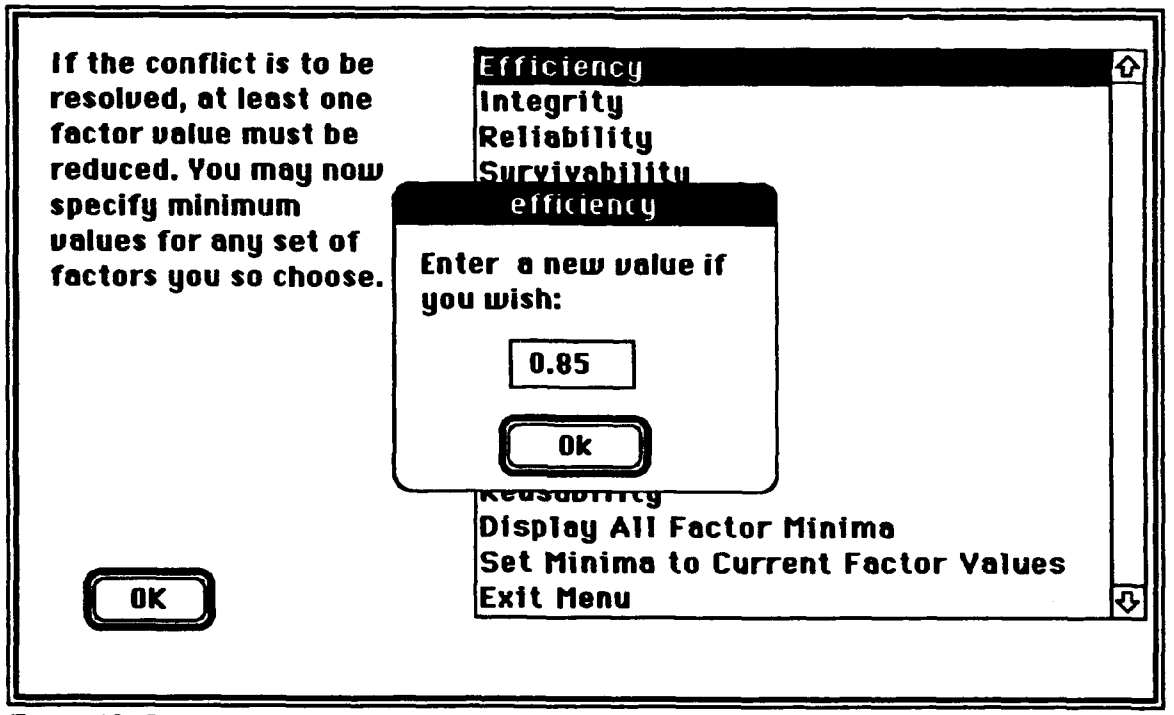


Figure 12. Factor goal modification.

This screen provides a capability to modify a factor goal. In this example, only the goal for efficiency is being lowered from the initial goal of 0.90 (excellent) to 0.85 (good).

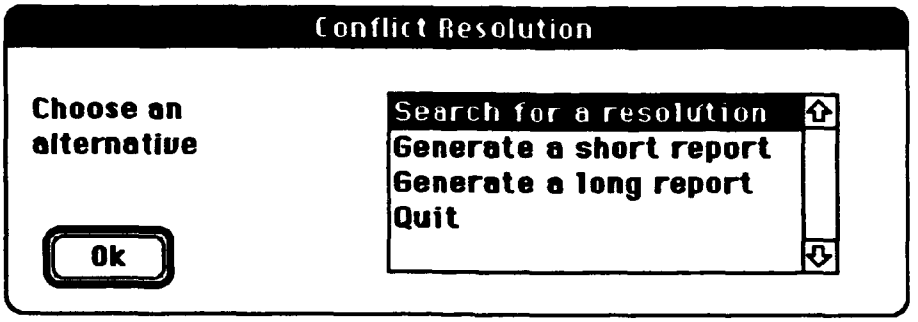


Figure 13. Search for a solution.

Reducing the goal for efficiency was not sufficient to resolve the goal conflicts. The goal set is still inconsistent; otherwise, the program would generate a screen indicating that there are no remaining conflicts and the goal set is consistent. The top menu selection instructs CORE to execute the solution search algorithms. Eventually, CORE will generate eight candidate solutions, all of which contain the acceptable value of 0.85 for the efficiency factor.

A solution has been found involving factor maintainability . Ok

The chart below shows factor values and permissible maxima after (1) factors have been set to the minimum values specified earlier, (2) the value of maintainability has been reduced as shown and (3) all factor interrelationships have been recomputed.

Factor	Value	Maximum
Correctness	0.89	1.00
Efficiency	0.85	0.86
Expandability	0.79	1.00
Flexibility	0.89	1.00
Integrity	0.90	0.98
Interoperability	0.80	1.00
Maintainability	0.88	1.00
Reliability	0.89	0.99
Usability	0.90	1.00
Verifiability	0.88	1.00

Figure 14. Solution 1.

CORE reports the first alternative: reduce the goal for Maintainability from 0.90 (excellent) to 0.88 (good). Program execution time was five seconds. As a result of interrelationships and shared criteria, other factor goals have been modified. Compare this goal set with the initial goal set shown in figure 2B.

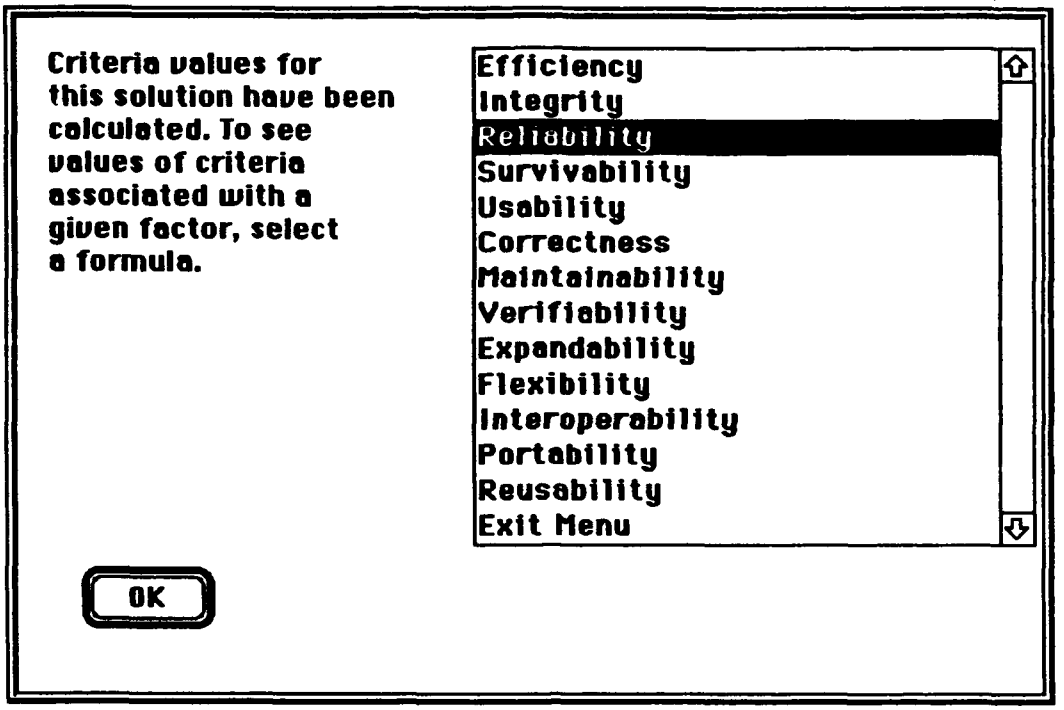


Figure 15. Underlying criteria values.

This screen provides a capability to review the criteria values factor/criteria values associated with candidate solutions.

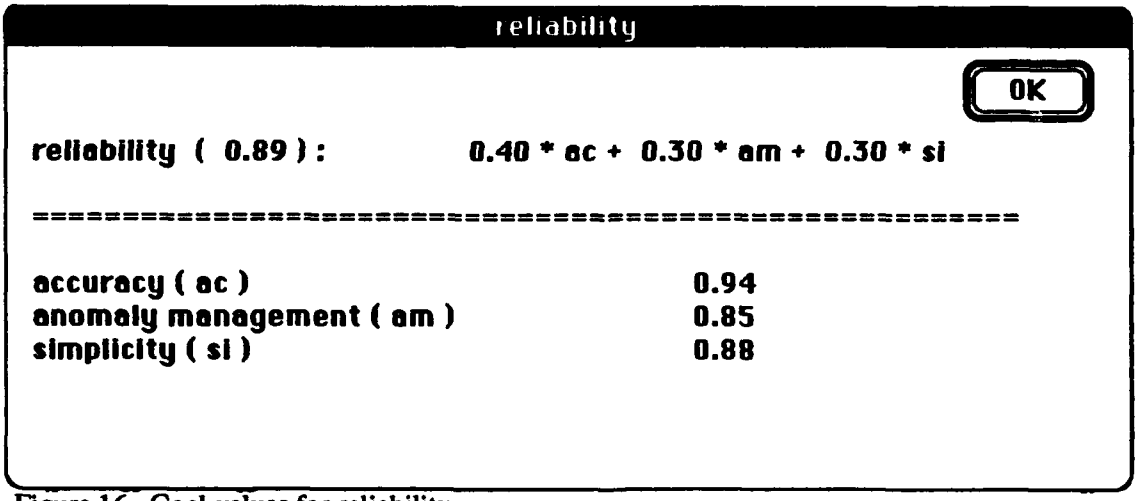


Figure 16. Goal values for reliability.

Criteria level information can support the decision making process during comparison of candidate solutions. This screen shows that the goal for reliability, compared to figure 8B, has been raised from average to good, due primarily to an increase in the target value for accuracy from 0.60 to 0.94.

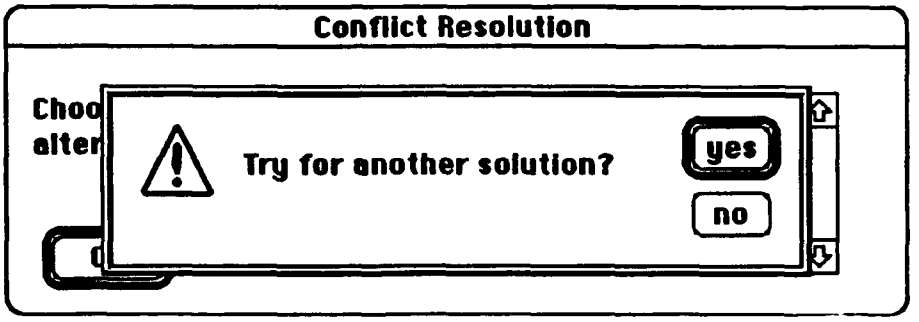


Figure 17. Another solution?

CORE indicates an additional solution(s) may be available.

A solution has been found involving factor verifiability .

Ok

The chart below shows factor values and permissible maxima after (1) factors have been set to the minimum values specified earlier, (2) the value of verifiability has been reduced as shown and (3) all factor interrelationships have been recomputed.

Factor	Value	Maximum
Correctness	0.90	1.00
Efficiency	0.85	0.86
Expandability	0.79	1.00
Flexibility	0.89	1.00
Integrity	0.90	0.98
Interoperability	0.80	1.00
Maintainability	0.88	1.00
Reliability	0.89	0.99
Usability	0.90	1.00
Verifiability	0.88	1.00

Figure 18. Second solution.

CORE displays a second solution: reduce the target goal for verifiability from excellent to good. Execution time is 1.5 seconds.

A solution has been found involving factor expandability .

Ok

The chart below shows factor values and permissible maxima after (1) factors have been set to the minimum values specified earlier, (2) the value of expandability has been reduced as shown and (3) all factor interrelationships have been recomputed.

Factor	Value	Maximum
Correctness	0.90	1.00
Efficiency	0.85	0.86
Expandability	0.78	1.00
Flexibility	0.88	1.00
Integrity	0.90	0.98
Interoperability	0.80	1.00
Maintainability	0.89	1.00
Reliability	0.89	0.99
Usability	0.90	1.00
Verifiability	0.88	1.00

Figure 19. Third solution. (t=2 seconds).

CORE displays a third solution: reduce the target goal for expandability from good to average.

A solution has been found involving factor flexibility .

Ok

The chart below shows factor values and permissible maxima after (1) factors have been set to the minimum values specified earlier, (2) the value of flexibility has been reduced as shown and (3) all factor interrelationships have been recomputed.

Factor	Value	Maximum
Correctness	0.90	1.00
Efficiency	0.85	0.86
Expandability	0.78	1.00
Flexibility	0.88	1.00
Integrity	0.90	0.98
Interoperability	0.80	1.00
Maintainability	0.89	1.00
Reliability	0.89	0.99
Usability	0.90	1.00
Verifiability	0.89	1.00

Figure 20. Fourth solution. (t=2 seconds).

CORE displays a fourth solution: reduce the target goal for flexibility from excellent to good.

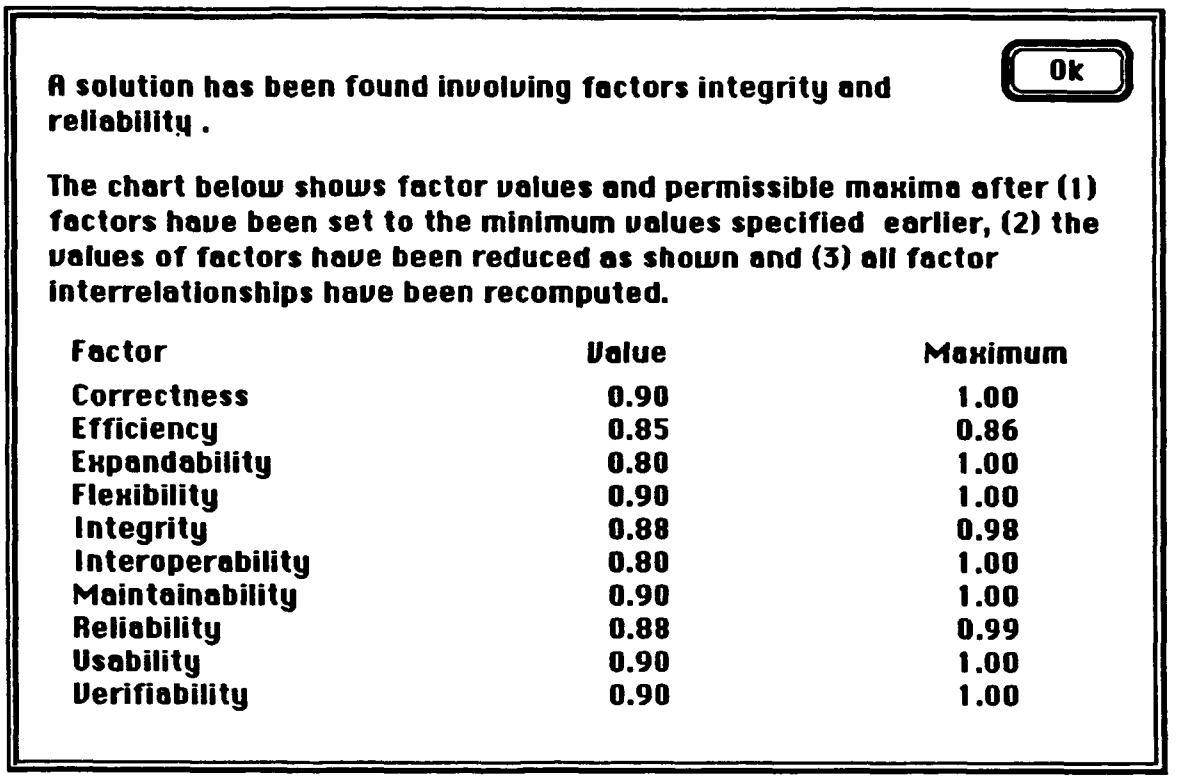


Figure 21. Fifth solution. (t=3 seconds).
CORE displays a fifth solution: reduce both integrity and reliability from excellent to good.

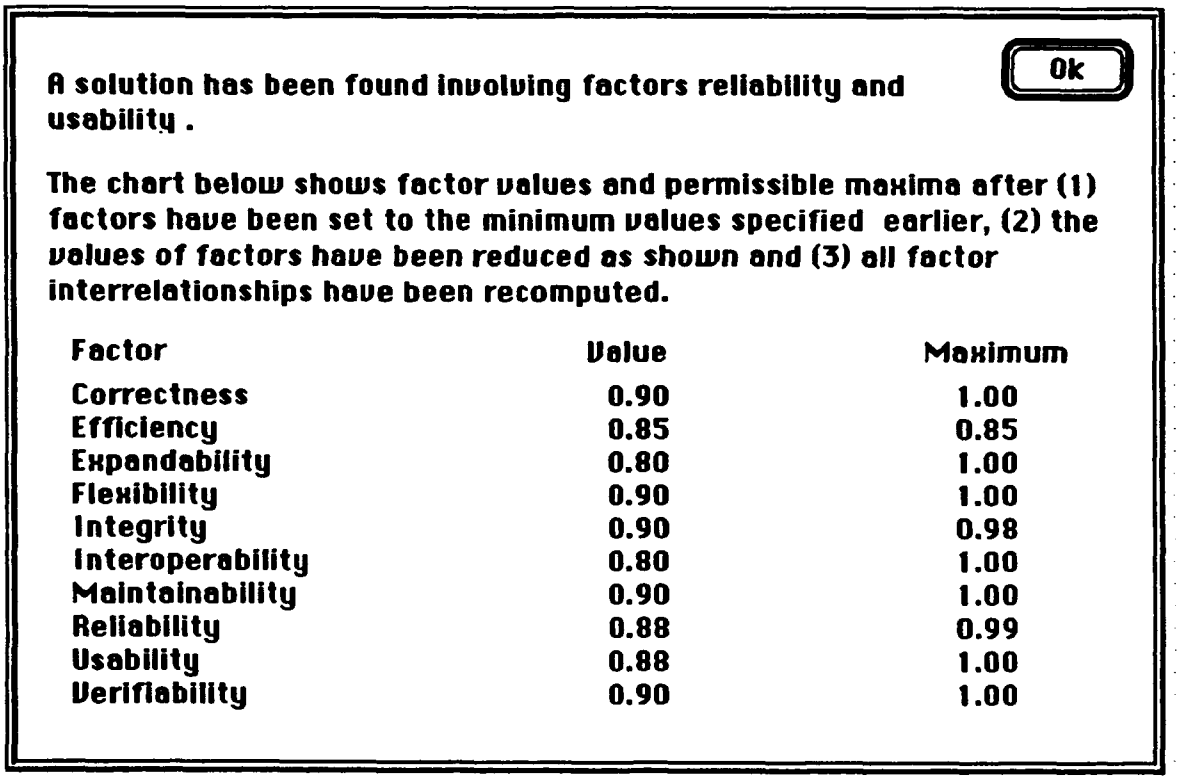


Figure 22. Sixth solution. (t=4 seconds).
CORE displays a sixth solution: reduce both reliability and usability from excellent to good.

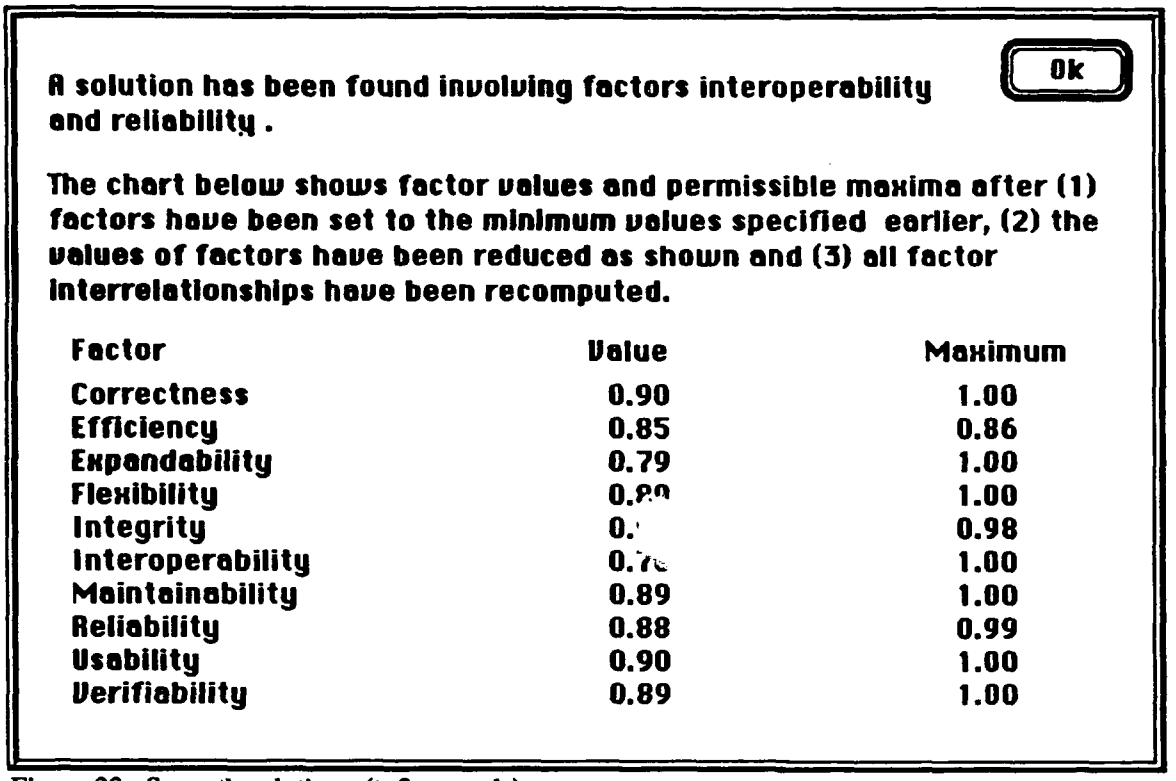


Figure 23. Seventh solution. (t=3 seconds).
CORE displays a seventh solution: reduce reliability to good and interoperability to average.

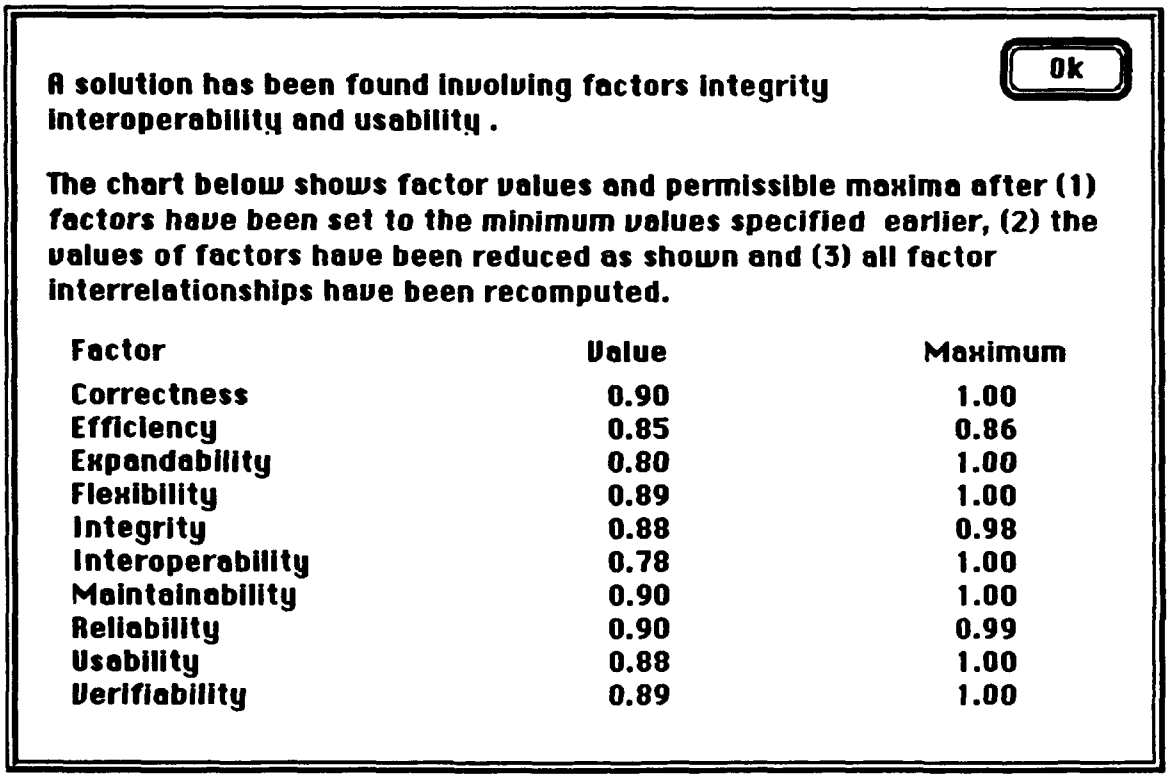


Figure 24. Eighth solution. (t=5 seconds).
CORE displays an eighth solution: reduce both integrity and usability from excellent to good, and reduce interoperability from good to average.

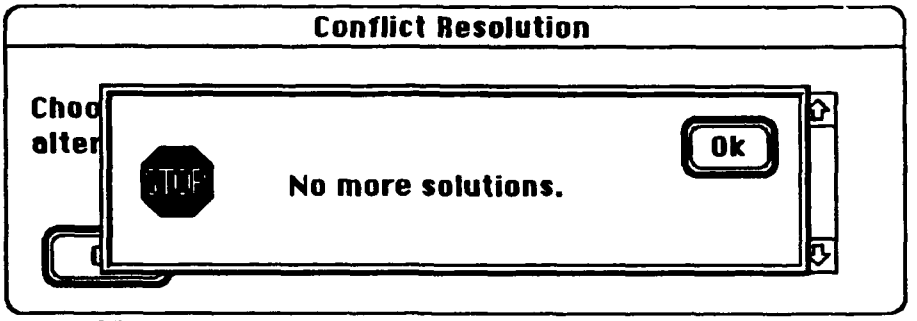


Figure 25. No more solutions. (t=15 seconds).

CORE indicates that there are no more solution candidates based on the initial starting conditions.

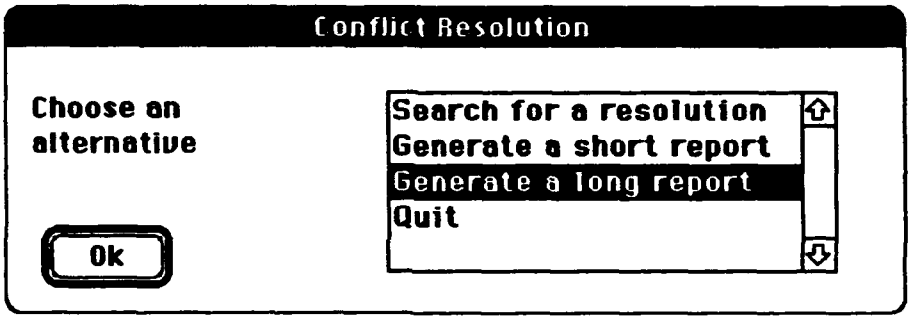


Figure 26. Reporting options.

This screen provides a capability to request a summary report of all solutions generated at two levels of detail.

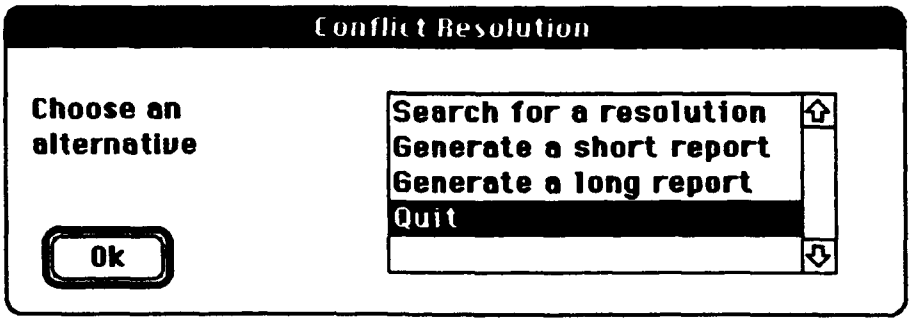


Figure 27. Quit CORE.

```
                    ==========================
                    CONFLICT RESOLUTION REPORT
                    ==========================

================================================================
Snapshot of the data base at the time the conflict was discovered,
        and just after the user set quality factor minima.
================================================================

                    =====================================
                          Quality Factors and Values
                    =====================================

Factor                  Value                   Maximum

correctness             0.90                    1.00
efficiency              0.85                    0.78
expandability           0.80                    1.00
flexibility             0.90                    1.00
integrity               0.90                    0.97
interoperability        0.80                    1.00
maintainability         0.90                    1.00
reliability             0.90                    0.98
usability               0.90                    1.00
verifiability           0.90                    1.00

                    =====================================
                          Quality Factors and Formulas
                    =====================================

correctness       =    0.40*cp + 0.30*cs + 0.30*tc
efficiency        =    0.10*ec + 0.80*ep + 0.10*es
expandability     =    0.20*at + 0.20*ge + 0.00*vr + 0.20*mo +
                       0.20*sd + 0.20*si
flexibility       =    0.25*ge + 0.25*mo + 0.25*sd + 0.25*si
integrity         =    1.00*ss
interoperability  =    0.20*cl + 0.20*fo + 0.20*id + 0.20*sy +
                       0.20*mo
maintainability   =    0.20*cs + 0.20*vs + 0.20*mo + 0.20*sd +
                       0.20*si
reliability       =    0.40*ac + 0.30*am + 0.30*si
usability         =    0.50*op + 0.50*tn
verifiability     =    0.25*vs + 0.25*mo + 0.25*sd + 0.25*si
```

Shared Criteria, Acronyms and Values

Criterion	Acronym	Value
anomaly management	am	0.85
consistency	cs	0.90
visibility	vs	0.90
generality	ge	0.90
independence	id	0.85
modularity	mo	0.90
self descriptiveness	sd	0.90
simplicity	si	0.90

Unique Criteria, Acronyms and Values

Criterion	Acronym	Value
accuracy	ac	0.94
autonomy	au	0.00
distributedness	di	0.00
effectiveness-communication	ec	0.85
effectiveness-processing	ep	0.85
effectiveness-storage	es	0.85
operability	op	0.90
reconfigurability	re	0.00
system accessibility	ss	0.90
training	tn	0.90
completeness	cp	0.90
traceability	tc	0.90
application independence	ap	0.00
augmentability	at	0.40
commonality	cl	0.75
document accessibility	do	0.00
functional overlap	fo	0.75
functional scope	fs	0.00
system clarity	st	0.00
system compatibility	sy	0.75
virtuality	vr	0.40

==========
The following solutions were discovered.
==========

Solution 1

Lower the value of maintainability to Good.
After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.89	1.00
reliability	0.89	0.99
efficiency	0.85	0.86
integrity	0.90	0.98
usability	0.90	1.00
maintainability	0.88	1.00
verifiability	0.88	1.00
expandability	0.79	1.00
flexibility	0.89	1.00
interoperability	0.80	1.00

==========
Solution 2
==========

Lower the value of verifiability to Good.
After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.89	0.99
efficiency	0.85	0.86
integrity	0.90	0.98
usability	0.90	1.00
maintainability	0.88	1.00
verifiability	0.88	1.00
expandability	0.79	1.00
flexibility	0.89	1.00
interoperability	0.80	1.00

==========
Solution 3
==========

Lower the value of expandability to Average.
After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.89	0.99
efficiency	0.85	0.86
integrity	0.90	0.98
usability	0.90	1.00
maintainability	0.89	1.00
verifiability	0.88	1.00
expandability	0.78	1.00
flexibility	0.88	1.00
interoperability	0.80	1.00

==========
Solution 4
==========

Lower the value of flexibility to Good.
After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.89	0.99
efficiency	0.85	0.86
integrity	0.90	0.98
usability	0.90	1.00
maintainability	0.89	1.00
verifiability	0.89	1.00
expandability	0.78	1.00
flexibility	0.88	1.00
interoperability	0.80	1.00

==========
Solution 5
==========

Lower the values of
integrity to Good,
reliability to Good

After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.88	0.99
efficiency	0.85	0.86
integrity	0.88	0.98
usability	0.90	1.00
maintainability	0.90	1.00
verifiability	0.90	1.00
expandability	0.80	1.00
flexibility	0.90	1.00
interoperability	0.80	1.00

==========
Solution 6
==========

Lower the values of
reliability to Good,
usability to Good

After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.88	0.99
efficiency	0.85	0.85
integrity	0.90	0.98
usability	0.88	1.00
maintainability	0.90	1.00
verifiability	0.90	1.00
expandability	0.80	1.00
flexibility	0.90	1.00
interoperability	0.80	1.00

==========
Solution 7
==========

Lower the values of
interoperability to Average,
reliability to Good

After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.88	0.99
efficiency	0.85	0.86
integrity	0.90	0.98
usability	0.90	1.00
maintainability	0.89	1.00
verifiability	0.89	1.00
expandability	0.79	1.00
flexibility	0.89	1.00
interoperability	0.78	1.00

==========
Solution 8
==========

Lower the values of
integrity to Good,
interoperability to Average,
usability to Good

After recalculating factor interrelationships, the data base will look like this.

Quality Factors and Values

Factor	Value	Maximum
correctness	0.90	1.00
reliability	0.90	0.99
efficiency	0.85	0.86
integrity	0.88	0.98
usability	0.88	1.00
maintainability	0.90	1.00
verifiability	0.89	1.00
expandability	0.80	1.00
flexibility	0.89	1.00
interoperability	0.78	1.00

==========
No more solutions were found.
==========

5 RECOMMENDED ENHANCEMENTS to CORE

CORE was developed as a standalone, proof-of-concept for extending the capabilities of the Assistant for Specifying the Quality of Software. During the informal testing of CORE on a range of examples, several priority enhancements were identified:

(1). Present information in a graphics format. Virtually all CORE output is now presented in textual and tabular formats. In general, determine good graphics representations effective for displaying trade off analysis information. Incorporate user-modifiable smart decision rules to provide additional support for evaluation of candidate solutions.

(2). Change the logic governing the fixed presentation of interface screens to provide a capability to backtrack to prior screens and thereby invoke CORE capabilities as suggested by intermediate results.

(3). Present information valuable to software developers. CORE processes at the criteria level. In the past, all emphasis on uses of the Software Quality Methodology have focused setting quality goals at the factor level. A given {quality goals} has implications for the development staff; i.e, they will be required to develop software which meets specified criteria scores. Inputs from the development staff can serve as additional information during the process of final {quality goals}determination.

(4). Implement smart search algorithms which will quickly eliminate infeasible solutions and will discover a greater number of solutions. In the current version, CORE finds candidate solutions to inconsistent goal factor sets by selectively reducing factor goals only once by a fixed amount equal to 0.02.

(5). Incorporate fuzzy logic to provide an enhanced capability to specify degrees of desired factor goal levels.

(6). Implement a capability to conduct cost impact studies of the alternatives generated in the technical feasibility stage.

(7). Review the current literature on multiobjective/multiattribute optimization theory. Evaluate recent algorithm and concept developments for possible use in CORE.

SECTION IV

REFERENCES

[1] Jeffrey A. Lasky and Alan R. Kaminsky, and Wade Boaz, "Software Quality Measurement Methodology Enhancements Study Results, "Final Technical Report, RL-TR-89-317, January 1990.

[2] Jeffrey A. Lasky and Michael J. Lutz, "Software Quality Methodology Integration Study Results, Final Technical Report, RL-TR-92-79, May, 1992.

[3] Larry Kahn and Steve Keller, "The Assistant for Specifying the Quality of Software (ASQS) Operational Concept Document, RADC-TR-90-195 Vol. I (of two), Sept, 1990.

[4] T.P. Bowen , G. B. Wigle, and J. T. Tsai, "Specification of Software Quality Attributes", Volumes I, II, and III, RADC-TR-85-37, February, 1985.

[5] Ruben Prieto-Diaz and Guillermo Arango (eds.), *Domain Analysis and Software Systems Modeling*, IEEE Computer Society Press, 1991.

[6] Douglas Schaus, "Assistant for Specifying the Quality of Software (ASQS), Mission Area Analysis, RADC-TR-90-348, December 1990.

[7] Tarek Abdel-Hamid and Stuart E. Madnick, *Software Project Dynamics: An Integrated Approach*, Prentice-Hall, 1991.

[8] Chien-Ching Cho, "A System Dynamics Model of the Software Development Process," MTR-10588, MITRE Corporation, May, 1989.

[9] Herb Krasner, et.al., "Lessons Learned from a Software Process Modeling System," *Communications of the ACM*, (35,9), September 1992, pp. 91-100.

[10] Bill Curtis, Marc I. Kellner and Jim Over, "Process Modeling," *Communications of the ACM*, (35,9), September 1992, pp. 75-90.

[11] R.H. Cobb, A. Kouchakdjian, and H.D. Mills, "The Cleanroom Engineering software development process", IBM STARS IR70/E CDRL 7001, by Software Engineering Technology, Vero Beach FL and IBM Federal Sector Company, Gaithersburg, MD, February 28, 1991.

[12] R. A. Radice, et.al., "A Program Process Architecture," *IBM Systems Journal*, (24,2), 1985, pp. 79-90.

[13] Patricia Pierce, Richard Hartley, and Sullen Worrells, "Software Quality Measurement Demonstration Project II", RADC-TR-87-164, October, 1987.

[14] Larry Kahn and Steve Keller, "The Assistant for Specifying the Quality of Software (ASQS) User's Manual, RADC-TR-90-195 Vol II (of two), Sept, 1990.

*U.S. GOVERNMENT PRINTING OFFICE: 1993-710-093-

MISSION

OF

ROME LABORATORY

Rome Laboratory plans and executes an interdisciplinary program in research, development, test, and technology transition in support of Air Force Command, Control, Communications and Intelligence (C^3I) activities for all Air Force platforms. It also executes selected acquisition programs in several areas of expertise. Technical and engineering support within areas of competence is provided to ESD Program Offices (POs) and other ESD elements to perform effective acquisition of C^3I systems. In addition, Rome Laboratory's technology supports other AFSC Product Divisions, the Air Force user community, and other DOD and non-DOD agencies. Rome Laboratory maintains technical competence and research programs in areas including, but not limited to, communications, command and control, battle management, intelligence information processing, computational sciences and software producibility, wide area surveillance/sensors, signal processing, solid state sciences, photonics, electromagnetic technology, superconductivity, and electronic reliability/maintainability and testability.