Simulations for very early lifecycle quality evaluations

Eliza Chiang[1], Tim Menzies[2]

[1]Department of Electrical & Computer Engineering,

University of British Columbia, 2356 Main Mall, Vancouver, BC, Canada

http://www.ece.ubc.ca/~elizac/vio/index.html; echiang@interchange.ubc.ca;

[2]Lane Department of Computer Science,

West Virginia University, PO Box 6109, Morgantown, WV, 26506-6109, USA

http://tim.menzies.com; tim@menzies.com

Summary

Chung et al. have proposed a graphical model that captures the inter-dependencies between design alternatives in terms of synergy and tradeoffs. This model can assist in identifying quality/risk trade-offs early in the life cycle of software development, such as architectural design and testing process choices. The Chung et.al. Method is an analysis framework only: their technique does not include an execution or analysis module. This paper presents a simulation tool developed to analyze such a model, and techniques to facilitate decision making by reducing the space of options worth considering. Our techniques combine Monte Carlo simulations to generate options with a machine learner to determine which option yields the most/least favorable outcome. Experiments based on the above methodology were performed on two case studies, and the results showed that treatment learning successfully pinpointed the key attributes among uncertainties in our test domains.

Keywords: Modeling Methodology, Software Process Models, Requirement Engineering, Software Quality Assurance, Monte Carlo Simulation

I. Introduction

Software system must meet all the functional requirements in order to provide desired functionalities to users. In addition, it must exhibit extra *non-functional* software quality attributes such as accuracy, security, performance, and other business goals. As there are no

clear-cut criteria to determine whether these goals are satisfied, *Chung, Nixon, Yu and Mylopoulous* ([Chung99]) used the notion of *softgoals* to represent such goals. *Chung et. al.* also define an entire softgoal modeling framework, featuring tradeoffs and inter-dependencies between system quality attributes and design alternatives. But their framework is a paper design only: if an analyst wants to simulate a softgoal system, they face the problem of simulations across a space of uncertainties intrinsic to softgoals. For example, an analyst can connect two softgoals and say (e.g.) *softgoal1 helps softgoal2* where "*helps*" is the second strongest of the four qualitative influences defined in a softgoal framework[1]. Analysts find it intuitive to specify their connection in such a simple qualitative format. However, qualitative influences (such as "*helps*") are subjected to individuals' beliefs, and are thus prone to be inconsistent. Our goal, therefore, is to develop a simulation tool that finds stable conclusions across inconsistencies within a softgoal framework.

Aside from inconsistent beliefs, another problem with drawing conclusions from a softgoal framework is the lack of supportive data. In the current software engineering practice, there is not much data available to perform statistical analysis on them ([Me01]). This is especially true during the early lifecycle of software development, when decisions are made based on uncertain and subjective knowledge. And in the case of advanced technologies and systems, there is little past experience to learn from. Without supportive data, the relevance of any conclusion drawn from a softgoal framework is questionable. In spite of this, estimations on the potential risks and benefits of design decisions during the earlier requirement phase is essential, because these early decisions have the most leverage to influence the development to follow. The *Softgoal Simulation Tool* presented in this paper, therefore, is designed to aid decision making in times such as early software development lifecycle, a time when domain knowledge is incomplete and inconsistent.

[1] The qualitative influences defined by Chung et al ([Chung99]) are "MAKE", "HELP", "HURT" and "BREAK".

The premise of our methodology is that within a large space of uncertainties generated from a model, there often exist emergent stable properties ([Me02]). If isolated, these properties can be used to drive a system towards the more/less preferred direction. In order to find such consistent behaviors, we apply "bounded" randomness (i.e. guesses that fall within some defined range) to handle imprecise knowledge, and utilize *Monte Carlo* simulation ([Kalos86]) to explore a wide range of system behaviors. This generates a large range of behaviors which must be analyzed. TAR2 treatment learner, an analytic tool developed by Menzies and Hu ([Meh01]), is employed to automatically summarize these behaviors, and return recommendations that can drive the system to some preferred mode. For example, Feather and Menzies ([Feath2]) describe one application that used formal requirements model written at *NASA Jet Propulsion Laboratory (JPL)* ([JPL]) for deep space satellite design. The formal model could generate a cost and a benefits figure for each possible configuration of the satellite (some 10^{30} options in all). The black dots at the top figure in figure 1 shows what happens after 30,000 Monte Carlo simulations of that model: note the very wide range of cost and benefits. After treatment learning, a small number of constraints on the satellite configurations were found that, after 30,000 more Monte Carlo simulations, yielded the black dots at the bottom figure of figure 1. There are two important features of these black dots at the bottom figure. Firstly, compared to the initial black dots (shown in the top figure), the variance in the costs and benefits is greatly reduced. Secondly, the mean value of the costs and benefits are improved; i.e. reduced cost and higher benefits. The success of the Feather & Menzies application lead to the speculation that one might understand the space of options within softgoals via *Monte Carlo* simulation and treatment learning.

As for the implementation of the *Softgoal Simulation Tool*, it is designed to be light-weight and highly customizable to different business goals. Our approach is somewhat different from the standard simulation methods in the software engineering or process modeling community. Standard methods include distributed agent-based simulations ([Clancey96]),

discrete-event simulation ([Harrell00, Kelton02, Law00])[2], continuous simulation (also called system dynamics) ([Abdel-Hamid91, Sterman00]), state-based simulation (which includes Petri Net and data flow approaches) ([Akhavi93, Harel90, Martin00]), our methods are closer to logic-based ([Bratko01, chapter 20]), or rule-based simulations ([Mi90]). In our approach:

- A model is defined that is a set of logical constraints between variables.
- A solution is generated from that model that satisfies those constraints.

In the ideal case, all model constraints can be satisfied. However, in the case of models generated from competing stakeholders, this may not be possible. Hence, our approach offers a range of operators which tries to satisfying all, many, or one of a set of constraints. The appropriate selection of operators depends on the business at hand. In the case of software quality assurance, for example, one might combine all softgoals within the framework with logic ANDs to model the strictest quality assurance scheme[3], or with logic ORs for the loosest. Users may also use a combination of the available operators to create framework that most resemble the actual system. Examples on how these operators can be configured to suit individual business needs are presented in the case studies section (section III and IV).

The rest of this paper is organized as follows: using the *Keyword in Context (KWIC)* framework ([Shaw96]) as an instructional example, we first introduce the Softgoal framework model proposed by *Chung et al*. Second, we present the inference process adopted by our *Softgoal Simulation Tool* to execute such model. Third, we explain how *Monte Carlo* simulation and TAR2 treatment learning are coupled to pinpoint consistent properties within the softgoal framework, properties that can drive the system toward some preferred state. Two case studies are presented later in this paper: the first one is the analysis of the *KWIC* framework, a small example which is discussed comprehensively to illustrate

[2] See also the http://imaginethatinc.com web site

[3] Some frameworks may not yield any relevant conclusion when inferred under the strictest constraint. It is because frameworks may compose of softgoals that benefit some while harm other softgoals, and this may prevent the inference engine to draw any conclusion if a majority of softgoals need to be satisficed.

our proposed technique; then, we introduce an advance satellite design project (SR-1) taken from *NASA IV&V Facility* as our second example. This example demonstrates how our method can be scaled up to real-world business use. Experimental results of both case studies and their implications are also presented. Finally, we summarize the proposed simulation technique and points to directions for future research.

II. Softgoal Modeling and Simulation

In this section, we outline a novel approach for system modeling and simulation. This approach involves the softgoal framework model, *Monte Carlo* simulation, and treatment learning ([Meh01]) to analyse and identify influential properties in the modelled system.

II.A. Softgoal framework: an Overview

Softgoal framework consists of three types of softgoals: the *Non-Functional-Requirement (NFR) softgoals*, the *operationalizing softgoals*, and the *claim softgoals*. *NFR softgoals* represent quality requirements such as "time-performance". *Operationalizing softgoals* comprise of possible solutions or design alternatives to achieve the *NFR softgoals* (e.g. "incorporate javascript in online storefront"). *Claim softgoals* argue the rationale and explain the context for a softgoal or interdependency link (e.g. a claim may argue that "client-side scripting loads faster"). As there are no clear cut criteria for success, *NFR softgoals* may not be absolutely achieved, yet they can be sufficiently satisficed[4]([Simon57]).

NFR softgoals can have an associated *priority*. *Priority softgoals* are shown in a softgoal *framework* with exclamation marks ("`!`", "`!!`"), or `critical`/`veryCritical` textually.

[4] Coined by H.A. Simon (United State social scientist and economist), "satisficed" is defined as to be satisficed with a minimum or merely satisfactory level of performance, profitability etc., rather than a maximum or optimum level. In the context of the softgoal framework, a softgoal is said to be satisficed when it is achieved not absolutely but within acceptable limits.

Priority specifies how important a softgoal is to be fulfilled for the success of the system. *Contribution* represents the interdependency between softgoals, as well as the influence a (*claim*) softgoal has on an interdependency link[5]. Listed in an increasingly positive magnitude, these *contributions* include BREAK ("−−"), HURT ("−"), UNKNOWN ("?"), HELP ("+") and MAKE ("++"). *Contributions* can also be combined among multiple softgoals and/or interdependency links through logic operations such as AND and OR.

The following presents the *Keyword in Context (KWIC)* System, a well-known example on software architectural design, to illustrate how the softgoal framework models design alternatives and quality attributes[6].

The framework shown in figure 2 defines the tradeoffs among *NFR*s and the architectural design alternatives within the *KWIC* domain. The top-level *NFR softgoals* - *Comprehensibility*, *Modifiability*, *Performance*, *Reusability* - are the quality requirements to be satisficed. The design alternatives - *Shared Data*, *Abstract Data Type*, *Implicit Invocation*, *Pipes and Filters*, populate the bottom-level as *operationalizing softgoals*. The sub-softgoals at the middle-level of the framework are obtained by decomposing the top-level *NFR softgoals*. In figure 2, for example, *Modifiability* considerations for a system are decomposed into concerns for *data representation*, *processes* and *functions*. The links from the *operationalizing softgoals* to *NFR softgoals* indicate the positive/negative impacts for each design alternatives had among quality factors. For instance, the *implicit invocation* regime makes an architectural design more extensible but requires more space, thus contributing to the corresponding softgoals ("*Extensibility[function]*" and "*SpacePerformance[system]*") respectively as illustrated in figure 2. Arguments such as "*expected size of data is huge*" is used to justify the statement: "*Pipe & Filter[Target*

[5] In graphical terms, they are the labels of the arrows between softgoals.

[6] The KWIC framework, taken from Chung et.al's book ([Chung99]), is a graphical expression of the architectural assessment knowledge from Shaw & Garlan ([Shaw96]).

System]" BREAKS(−−) "*Space Performance[System]*", and is represented by *Claim softgoal* ("*Claim [c4]*").

In addition, the "!!" symbol associated with the *NFR softgoal* "*Modifiability[Data Rep]*" indicates that it is a high priority quality attribute to be satisficed. Several other attributes, such as "*TimePerformance[System]*", "*Deletability[Function]*", "*Updatability[Function]*", also serve as critical factors for overall system quality.

The *KWIC* framework discussed above sets an example on how Softgoal Modeling technique can be applied to other systems to capture the tradeoffs/synergy between quality attributes and design alternatives. After a softgoal framework is constructed for the target system, it can be fed to the *Softgoal Simulation Tool* for automatic inference and simulation.

II.B. Inference

Once the Softgoal framework is defined, it is then encoded into text format for automatic inference[7]. The methodology for inferring the softgoal framework structure is described in this section.

Each Softgoal framework requires a top-level softgoal node that represents an abstraction of overall quality. Each search performed on the framework generates a consistent "world" - a scenario where the top-level softgoal is satisficed when a set of softgoals are satisficed / denied. This "world" can be different for each search, depending on the topology and randomness embedded in the framework definition. After a "world" is generated, its "goodness" is rated by computing the "benefit" and "cost" of this particular "world" based on various user-configured parameters.

[7] See site http://www.ece.ubc.ca/~elizac/vio/papers.html for framework encoding scheme and keyword definitions.

When uncertain qualitative inferences (e.g. HELP, MAKE) meet, combination logic is required to sum these influences. Logic operators supported by the *Softgoal Simulation Tool* include AND, OR, and ANY. Chaining softgoals with AND imposes the strictest constraint towards satisficing their parent softgoals, whereas chaining with OR requires only one satisficed softgoal to satisfice its parent. Logic ANY is similar to OR in its satisficing criteria, except that the inference engine would try to prove more than one of the chained softgoals[8]. Different business concerns on a domain can be addressed by using different combinations of logical operators in the analysis of softgoal frameworks. We constructed two sample applications, the Rigorous Quality Assurance and the Weak Quality Assurance Scheme, in our study of the KWIC framework to demonstrate such capability. Details and experimental results are presented in section III.

II.C. Calculation of Cost and Benefit

As mentioned in the previous section, each inference on the framework results in a cost and benefit score. To compute these values, qualitative factors such as softgoal *priorities* ("!", "!!") and *contributions* ("-, ++") are involved. Numerical values are required to represent these factors during automatic inference, yet there is no definition available for quantification. This section outlines the approach taken by our simulation tool to handle the calculations of benefit and cost under such limitation.

II.C.1. Handling Source of Uncertainty within Softgoals

As there is no conventional basis for quantifying subjective knowledge (e.g. HELP, MAKE), a quantification rule is thus created to state the rankings of various qualitative strengths. Under this rule, the *mean* of all quantified parameters must satisfy some numerical

[8] For implementation details, please see site:

http://www.ece.ubc.ca/~elizac/vio/softgoal/logic_operations_used_in_framework_inference.html

constraints. A typical quantification rule is stated below:

```
0 <= score("--") <= score("-") <= 1 <= score("+") <=
                    score("++") <= 2
```

A score less than 1 reflects a weakening effect. Having fixed a (0..1) range for weakening, a range between 1 and 2 is used to reflect strengthening of the influence.

Each score is expressed as a Gaussian with user-defined *mean* and *variance* values. *Variances* characterize the score distributions and determine how much the Gaussians overlap each other. Users can adjust the *variances* according to the business model and the degree of inconsistency of domain knowledge[9].

Cost of each softgoal can be configured as either a static or random value. We implemented the costs of all the design alternatives in the *KWIC* framework (section III) to be static. Randomized cost calculation is applied to the framework of the SR-1 Project (section IV).

As mentioned before, the cost and benefit computed in each inference produce a "rating" on the "desirability" of a particular "world". These ratings, once obtained by performing *Monte Carlo* process, are used by a treatment learner for classification. Details on treatment learning are given in the next section.

II.D. Treatment Learning with TAR2

After the ratings and the corresponding behaviors of the "worlds" are recorded, we apply treatment learning to summarize this data. To allow TAR2 treatment learner to classify each "world" according to its cost and benefit score, a classification and ranking scheme is required to map ranges of costs/benefits to appropriate categories. For the case study of the *KWIC* framework, each "world" is rated based on the following classification scheme: the

[9] For more details on benefit/cost implementation, please see site: http://www.ece.ubc.ca/~elizac/vio/softgoal/benefit_and_cost_calculation.html

ranges of benefit were sub-divided into six bands - *vvlow*, *vlow*, *low*, *high*, *vhigh* and *vvhigh* (in increasing magnitude), whereas cost is sub-divided by its discrete values (from 0 to 5). Each band has roughly the same number of samples. Combining each band of the cost and benefit yields 36 classes (see table I for the ranking function described). This classification scheme is applied to both rigorous (figure 3) and loose (figure 4) quality assurance on *KWIC* framework. This scheme takes account on both benefit and cost with slight preference towards lower cost. For example, *Cost=zero,Benefit=vhigh* has a higher ranking than *Cost=one,Benefit=vvhigh*. Different preference scheme can be configured for specific business concerns.

For our *Softgoal Simulation tool*, TAR2 treatment learner ([Meh01]) is used to perform data analysis. Base on the class ranking (table I), TAR2 searches the datasets for the candidate attribute ranges, that is, ranges that are more common in the highly ranked classes than the other classes. In the *KWIC* domain, such a candidate is a range of design approaches that drive the system into high quality/low cost, or low quality/high cost if the range of undesirable design options is of concern. Knowing this range of attributes can greatly assist in making design decisions, as the space of considerations is narrowed down to only the attributes that would assert positive/negative impacts towards the system.

Prior to the discussion on the case studies in the next section, we introduce the **incremental treatment learning** ([Me02]) strategy that is used in our study of the *KWIC* framework. To apply incremental treatment learning, a *Monte Carlo* simulator executes and generates datasets on the softgoal framework model. TAR2 condenses this dataset to a set of proposed treatments. After some discussions, users add the approved treatments as constraints for another round of *Monte Carlo* simulation. This cycle repeats until users see no further improvement.

The next section presents two case studies where our proposed simulation technique is

applied.

III. Experiments and Result on the KWIC system

As detailed in section II, the *KWIC* framework models the architectural design alternatives and quality attributes that are of business concern. The objective of this experiment is to look for design alternatives that would significantly impact the *KWIC* system among inconsistent knowledge such as trade-offs and beliefs.

In this section, we present two different logical interpretations applied onto the framework topology with respect to different business concerns. The rationale behind these interpretations and our observations are also discussed.

III.A. Experiment 1: Rigorous Quality Assurance

This experiment is intended for system designs where strict quality assurance is mandatory. The goal is to find out what design alternatives would optimize system quality attributes. As shown in figure 4, the *NFR softgoals* are combined with logic AND, meaning that all the softgoals have to be satisficed in order to satisfice their parent softgoal. To satisfice the *NFR softgoals* immediately above, the inference would try to satisfy the *operationalizing softgoals* as many as possible, and hence they are combined with logic ANY. The top-level *NFR softgoals* below the overall goal "*goodness of system"* are combined with logic ANY[10].

This above schematic is applied to the *KWIC* framework, and its implications is explained as follows: the *operationalizing softgoals*, namely the *sharedData[targetSystem], abstractDatatype[targetSystem], implicitInvocation[targetSystem],* and

[10] ANY is used instead of AND to allow proper inference on this particular framework, as it is impossible to satisfice all the desired system quality attributes represented by the top-level nfr softgoals.

pipe&Filter[targetSystem], are attached to *modifiability[DataRep]* with ANY, meaning that the inference engine will try to prove as many of the *operationalizing softgoals* as it can to satisfice the *modifiability[DataRep]*. Satisficing *modifiability[System]* means all its precondition softgoals – *modifiability[Process], modifiability[DataRep]*, and *modifiability[Function]* – are satisficed, for they are combined with an AND. As it is impossible to find a "world" where the *comprehensibility[System], modifiability[System], performance[System]* and *reusability[System]* are satisficed at the same time, they are chained with ANY (instead of AND) so that the top-level goal "*goodness[System]"* can be satisficed.

Calculations on benefits and costs, as well as other parameters, are summarized in figure 3. Notice that *deletability[System]* in figure 2 does not associate with any *operationalizing softgoal*. Thus, it is assumed that certain operation is performed for its fulfillment, and the cost of this unknown operation is equal to 1. The class ranking function is described in table I.

Results from incremental treatment learning of the *KWIC* framework using the rigorous quality assurance settings are summarized in table II, III, IV and V. In order to clearly show how TAR2 condenses data ranges to improve the mean of the more preferred class, the results for each of the incremental process are presented as percentile matrix. Each cell is colored on a scale ranging from white (0%) to black (100%).

As this experiment represents the case where business users put more focus on software quality (i.e. benefit scores) verses costs, the benefit improvement is emphasized in the following discussion on treatment results.

Table II shows resulting data ranges when no constraint was imposed on the architectural design options and claims. Table III, IV, V shows the result of applying incremental

treatments to figure 4. Note that as the key decisions accumulate, the variance in behavior decreased and the mean benefit scores improved. The mean benefit drifted from <5.5 before treatment (table II) to <11 at treatment round 4 (table V). Moreover, the number of samples fell into the high benefit ranges (<27.5 and <32) increased after treatment. Base on this result, developers may focus on key issues that would greatly impact overall software quality, such as whether or not to implement *shared Data* for the system. Alternatively, if in some dispute situation, an analyst could use *c2; c4; c5* as bargaining chips. Since these claims have little overall impact, our analyst could offer them in any configuration as part of some compromise deal in exchange for the other key decisions being endorsed.

As a last note on the result shown in table III, IV and V, we observed that cost increases as treatments accumulate. In other words, rigorous quality assurance costs the most but doubles the average benefit. With this new information, users are now informed enough to intelligently debate the merits of rigorous quality assurance over its cost.

III.B. Experiment 2: Weak Quality Assurance

Often, when outside consultants are called in to offer a rapid assessment on how to improve a problematic project, they seek the fewest actions that offer the most benefit. To handle this situation, we defined a variation of the *KWIC* framework to simulate a weaker form of quality assurance. This assurance scheme is a simple modification in terms of its logical operations (i.e. swapping logical operators between softgoals). As shown in figure 5, the *NFR softgoals* are combined with logic OR, meaning that the parent softgoal is satisficed when one of its contributing softgoals is satisficed. Similarly, only one of the *operationalizing softgoal* is needed to fulfill the satisficing criteria of the *NFR softgoal* immediately above. The parameter configuration for inference is the same as of rigorous quality assurance. In order to find the least preferred behavior, the class rankings are reversed as opposed to that of the rigorous quality assurance scheme. Incremental treatment learning is performed on figure 5,

and the results are shown in table VI to IX.

The goal of this experiment is to determine what would negatively impact software quality in the most liberal quality assurance scheme. Comparing table IX (after treatments) with table VI (before treatments), the number of samples fell into the lowest benefit range (<14.7) increased, which showed that benefit suffered as treatments accumulated. The results also suggest that, using the weaker form of quality assurance scheme, the overall software quality of the *KWIC* system suffers if *Pipe & Filter* is not implemented. Hence, users may center their discussions on the possibilities of implementing the *Pipe & Filter* option. Most importantly, high cost solutions can be avoided (note those results 35% over cost=3 in table IX) without degrading overall benefits.

We have presented the treatment learning results of the *KWIC* framework experiments on two distinct settings, and how these settings address different business concerns. Even though their implications are different, these experimental results demonstrated how TAR2 discovers a range of consistent behavior among the space of inconsistent information. Also, as incremental treatment is applied, *variance* is reduced and *mean* values of preferred classes improved.

IV. Case Study: NASA IV&V Activity Prioritization - A Study on the SR-1 Project

The case study discussed in section III demonstrated our simulation technique working on a small example, albeit one often cited in the literature. The SR-1 Project presents in this section shows how our *Softgoal Simulation Tool* scales up to modern real world software.

IV.A. The SR-1 Project

This case study demonstrates the capability of the *Softgoal Simulation Tool* to "learn" treatments from incomplete data. The overview and the goal of our study are outlined first, followed by details on the investigations and analysis process performed in this study. The treatment learning strategy is then defined, and result of our experimentations presented.

IV.A.1. Introduction

This section begins with some basic facts and terminologies being used throughout this case study. Then the objective of our study is defined, and the issues on data unavailability addressed.

The *NASA* SR-1 ([Sr102]) program refers to the technologies involved in advance satellite design. It is under a contract from *NASA*'s *Space Launch Initiative (SLI)*. In this study, we focus on the software components of this technology, specifically on a list of *Catastrophic/Critical/High Risk* (*CCHR*) functions and the standards used for evaluating their risk and criticality.

The *NASA IV&V Facility* is one of the organizations performing V&V on software projects such as SR-1. Verification & Validation (V&V) is a system engineering process employing a variety of software engineering methods, techniques, and tools for evaluating the correctness and quality of a software product throughout its life cycle ([Nasa]). Independent V&V (IV&V) is performed by organization that are technically, managerially, and financially independent of the development organization.

The *Criticality Analysis and Risk Assessment (CARA)* ([SR102]) process is a quantitative analysis used by the *NASA IV&V* personnel to determine the appropriate scope of V&V on a project. *CARA* is based on the notion that a function that has high criticality and high risk requires more extensive inspections than a function of lower criticality/risk. The *CARA*

analysis evaluates and rates the criticality and risk of software functions based on factors such as size and complexity of program code. These ratings are then used to calculate the *CARA* score for each of these functions. Appropriate *IV&V* resources are assigned based on these scores.

IV.A.2. Overview and Objective of this study

Like many other companies, project management at *NASA IV&V Facility* has to deal with business issues such as delivery deadlines and resource allocations. It is every manager's goal to optimize resource usage, reduce project costs while meeting deadline dates. On the other hand, each IV&V analysis activities consume different degree of resources, and some of these activities perform better in the V&V process than the others. Finding out which of these V&V activities are more powerful, and less costly at the same time, would be helpful for project resource management and task prioritization. The objective of our study, therefore, is to look for the analysis activities that are more cost-effective than others.

In our study of the SR-1 project, we applied the softgoal framework idea to sketch out the inter-dependencies between I*V&V* Analysis Activities and the Criticality/Risk assessments on SR-1 functions, which are summarized as follows:

- **Criticality** and **Risk Criteria**, such as *Performance and Operation*, are viewed as the quality attributes which each validated SR-1 functions are trying to satisfice. They are the *NFR softgoals* in the SR-1 framework.
- **SR-1 Software Functions** (e.g. *vehicle management*) are also viewed as *NFR softgoals*, as no software validation process can guarantee these functions to be absolutely flawless. Nonetheless, their correctness can be sufficiently satisficed by applying IV&V analysis activities.
- **IV&V Analysis Activities** serve as the *operationalizing softgoals* in the framework.
- **CARA Ratings** (*catastrophic*, *critical*, *high*, *moderate*, *low*) defines the impacts of

each function towards SR-1's overall criticality and risk factors upon failure. Thus, they become the inter-dependencies between **SR-1 Software Functions** and the **Criticality** and **Risk Criteria**.

- **Effectiveness of Analysis Activities** relates the **IV&V Analysis Activities** to the applicable **SR-1 Software Functions**.
- **Significance of SR-1 Functions** defines its *priority*.

To illustrate the above idea, a sample segment of the SR-1 framework is shown in figure 6. Figure 7 shows the criticality and risk ratings of the SR-1 functions, as well as their analysis levels resulting from the *CARA* process. The analysis activities these levels provide by *NASA* I*V&V* for requirements, design, code, and test are listed in figure 9. Each of these SR-1 functions maps to an analysis level, which is mapped to a set of activities assigned to analyze the function. For example, the function "*Target State Filter"* (*f[tFilter]*) is assessed to be level "*Limited"*(*L*), thus the analysis activities: *rav01 - rav09*, *dav01 - dav09*, *cav01 - cav06*, and *tav01 - tav07* are performed to validate the integrity of this function. Figure 8 shows the cost of these analysis activities in qualitative terms (*low*, *high* and *veryHigh*). The term "cost" is used to generalize factors such as time, manpower, and other I*V&V* resources.

As we proceeded on our analysis, we found that the SR-1 softgoal framework displayed typical features of real world systems - lack of domain knowledge and supporting data. First of all, we were unable to obtain any expert opinions regarding to the effectiveness of each analysis activities. Similarly, we have no information on which of the SR-1 function is more important than the others. Moreover, there is discrepancy in the scaling factors for cost calculations. These factors would defeat traditional quantitative approach in requirement analysis. With our proposed technique, however, we were able to perform inference and draw useful conclusion in spite of the lack of domain knowledge. The next session details how we handled the incomplete information in the SR-1 framework.

IV.B. Softgoal Framework Construction

The list below describes the uncertainty factors in the SR-1 framework, translated into softgoal framework-specific terminologies:

- The *contributions* of *operationalizing softgoals* (analysis activities) to *NFR softgoals* (SR-1 functions);
- The *priorities* of all *NFR softgoals*;
- The uncertainties intrinsic to the use of qualitative representations (e.g. "*catastrophic CARA*" ratings, "*veryHigh*" cost);

To allow inference while accounting for the above factors, we have made some assumptions and corresponding adjustments to the inference process[11]. They are listed as follows:

- Analysis activities will always contribute positively to the integrity of SR-1 functions; i.e. all the *operationalizing softgoals* will either HELP or MAKE their parent *NFR softgoals*. To comply with this assumption, each positive *contribution* is randomly chosen to be either HELP or MAKE by our *Softgoal Simulation Tool* during inference;
- Performing V&V on either `catastrophically_rated` or `critically/highly_rated` functions is assumed to be always beneficial towards the overall safety of SR-1 software. Also, we assumed that doing V&V to lowly-rated SR-1 functions has negative impacts on the framework. The rationale of such assumption is that the additional workload may hinder job performance of V&V specialists, and hence out-weighed the gains;
- For the `moderately_rated` functions, the effect is assumed to be either positive or negative. Therefore, such rating is transformed to be either `lowly_rated` or `critically_rated` during inference;
- All the *NFR softgoals* have the same *priorities*;

[11] For further details, please see http://www.ece.ubc.ca/~elizac/ vio/papers.html

- The numeric values of the qualitative terms fall within pre-defined ranges, as shown in figure 11;

Regarding to the cost discrepancy, two versions of cost functions were presented to us. Figure 10 describes these functions.

Other settings used in inferring SR-1 framework are shown in figure 11. The resulting SR-1 framework consists of 48 *operationalizing softgoal* nodes, 28 *NFR softgoal* nodes, and hundred of edges representing *softgoal contributions*. As this framework was too large to be legible, we were unable to present it in this paper.

After the SR-1 softgoal framework was constructed, we carried out our investigations using the *Softgoal Simulation Tool*. Details on the experimental settings (class ranking functions, logic configurations, etc) and treatment learning results are given in the next section.

IV.C. Experiments and Results

The class ranking function used for the SR-1 framework is similar to that of the KWIC framework. The range of benefits and costs were sub-divided into four bands (from *vlow* to *vhigh*), and each band consists of roughly the same examples. Table X shows these class rankings. Two studies were conducted based on this ranking function:

- In the "MOST PREFERRED" study, TAR2 looks for behaviors that would contribute to the integrity of the SR-1 functions.
- Conversely, in the "LEAST PREFERRED" study, we reversed the order of class ranks to find treatments, which would assert negative impact to the framework

We constructed and experimented on two variations of the SR-1 framework, which differed

in their logical compositions[12]. Figure 12 showed the weakest quality assurance scheme build to represents realistic business situation, where analysts try to perform as many analysis activities to fulfill the integrity of a SR-1 function, and hence the overall software quality. In this framework, the analysis activities at the bottom level were chained with an ANY, and attached to their corresponding SR-1 functions. The SR-1 functions were bind to each of its critical/risk criteria with an OR, which were also bind to their upper-level softgoals with OR. For instance, *rav01-rav09* and *tav01-tav07* are chained with ANY and attached under SR-1 function *f[cam]*, meaning that the associated activities would be proven as many as it could during inference to satisfice *f[cam]*. Function *f[cam]*, together with other functions (such as *f[vm], f[guid], f[nav]* etc.) were chained to the risk criteria *Ri[d]* with logic OR. Therefore, satisficing one SR-1 function would be sufficient to satisfice the risk criteria. Similarly, the risk criteria *Ri[a]* to *Ri[d]* are combined with OR under the overall *risk* softgoal, which was connected to the top-level softgoal with an ANY. In other words, satisficing one *risk* criteria would be enough to satisfice the overall *risk* softgoal, which would lead to the top-level softgoal being satisficed.

To compare with figure 12, we proposed another SR-1 framework to represent rigorous quality assurance. For this, we derived two prototypes of such a framework, presented in figure 13 and 14. Figure 14 defined the strictest form of quality assurance, in which all the *NFR softgoals* (i.e. SR-1 functions, risk/criticality criteria, *risk* and *criticality* softgoals, and the top-level softgoal) are combined with AND, except for the bottom level *operationalizing softgoals* (i.e. analysis activities), which were combined with ANY. We reviewed this configuration and found it corresponded to a "utopia" model of rigorous quality assurance, since it is impractical in real world situation to fulfill the complete set of quality requirements as implied by this model. Because of its lack of practical applications, we abandoned this model and experimented on a more "pragmatic" rigorous quality assurance model, as shown

[12] The appropriateness of these framework variants in representing the real situations has yet to be determined by NASA experts.

in figure 13. In this "pragmatic" model, ANY were used to replace AND in the "utopia" model as a weaker form of conjunctive logical operator. Further, OR was used to replace ANY to further relax the satisficing constraint. Under this scheme, one satisficed analysis activities at the bottom level of the framework would be sufficient to fulfill the SR-1 function softgoal at the upper level. The inference engine would attempt to satisfice these SR-1 function softgoals as many as possible to satisfice the *criticality*/*risk* criteria softgoals, which would also be attempted as many as possible for the upper level overall *criticality*/*risk* softgoals. Finally, the top-level softgoal would be satisficed when either one or both of the overall *criticality*/*risk* softgoals were satisficed. We found this model to be a closer match to the real world business case; hence it was used in performing the experiments in our studies.

After the framework variations were defined, Monte Carlo simulations were applied to each variant twice, each time with different cost functions. After that, treatment learning is applied to each set of data in finding the most/least favorable treatments. The effects of these treatments are compared with the control situations (i.e. no treatment) in terms of costs and benefits. Results from the experimentations described above are presented in two groups: table XI, XII, and XIII for weak quality assurance; and table XIV, XV, and XVI for "pragmatic" rigorous quality assurance.

Recall that the class ranking function defined in table X accounted on both benefit and cost with slight preference towards lower cost (e.g. *Cost=vlow, Benefit=high* has a higher ranking than *Cost=low, Benefit=vhigh*). Because of this setting, treatment learner would always recommend treatments that sacrifice a lower benefit for a lower cost. All out result sets reflected this particular class setting.

Several features of these results deserve comment. Consider the results of the experiment on SR-1 framework 1 (weak quality assurance): firstly, treatment eliminated samples within the *Cost=vlow, Benefit=vlow* range, from >34% before treatment to 0% after treatment.

Secondly, for the MORE PREFERRED system (where *tav09 of tal=y*), treatment learning drove the sample distributions towards a higher benefit range (*Benefit=high* and *Benefit=vhigh* occupied >76%, as opposed to <50% with no treatment). Third, the distributions of total benefit received after treatments were roughly the same for both MOST PREFERRED and LEAST PREFERRED system. However, the LEAST PREFERRED system (where *cav10 of cal=y*) suffered from very high cost (77% of the sample was classified as *Cost=vhigh*), compared to the MOST PREFERRED system (41% of the sample). This effect could be explained by the way the class ranking function was defined. Treatment learning for the MORE PREFERRED system would give recommendations that yield lower cost over lower benefit, whereas it would identify treatments that result in higher cost for the LEAST PREFERRED system.

The result for the "pragmatic" rigorous quality assurance (figure13) scheme is presented in a similar fashion as the weak one described above. The experimental data is shown in table XIV, XV and XVI. First of all, treatments for the MORE PREFERRED system (*dav12 of dal=n*) resulted in an increase of samples in the *Cost=vlow* and *Cost=low* range, from >50% to >69%. Nonetheless, the samples within the range *Benefit=vlow* and *Benefit=low* also increased (from 50% to <57%), a clear indication on the proportionality of cost and benefit. On the other hand, treatment for the LEAST PREFERRED system (*cav07 of cal=y*) resulted in very high cost (>85% in the *Cost=high* and *Cost=vhigh* range) compared to that of no treatment (<50%). However, the benefit did not significant increase correspondingly, as >52% in the *Benefit=high* and *Benefit=vhigh* range after treatment, where it was <50% before treatment.

To conclude the SR-1 case study, the following points summarize the results and our observations:

- The use of logic components significantly affects the treatments that TAR2 recommends.

- Variation on cost functions was found to have no observable effect to the resulting treatment recommendations from all SR-1 framework variants studied.
- For all experimentations on the SR-1 framework, TAR2's treatment recommendations have been 10-way cross-validated[13] and their trustworthiness ensured. In other words, all of TAR2's treatment recommendation remained stable, in spite of the uncertainties and imprecise factors (as discussed in section IV B) lay within the framework.
- For the results shown in table XV (on SR-1 framework 2), TAR2 suggested not to do *dav12* would be beneficial. Our treatment learning method can give advice on which activities not to be done in order to receive the most preferred outcome.

V. Related Work

Inference diagram ([Shachter86]) (a form of Bayes nets) has been used to sketch out subjective knowledge, then assess and tune each knowledge variable based on available data. This modeling scheme is used by *Burgess et al* ([Burgess01]) in evaluating requirements that are candidates to be included in the next release of some software. Bayesian reasoning theory adopts a quantitative, probabilistic approach of solving decision problem. Its algorithm is widely studied and well understood, and inference tool widely available. To apply Bayesian theory in defining the softgoal framework, one needs to translate the relationship between *operational softgoals* (design alternatives) and *NFR softgoals* (software quality attributes) into probabilities (e.g. 0.6, 0.8). These numerical values may not be as intuitive as linguistic descriptions (”HELP”, “MAKE”). Moreover, when data required to determine the likelihoods for the calculation of posterior probabilities is not available, certain assumption has to be made. These assumptions negate the authenticity of the answer obtained by Bayesian inference. Therefore, Inference Diagram may not be the best for early lifecycle requirement modeling.

[13] Cross-validation is a method of estimating generalization error based on “re-sampling”. In 10-way cross validation, the entire dataset is randomly split into 10 mutually exclusive subsets for approximately equal size. Each subset is being tested on the inducer that is trained by the other 9 subsets of data ([Kohavi95]).

Fuzzy Petri Nets, a formalism that combines fuzzy set theory and Petri Net theory, is a tool for the representation of uncertain knowledge about a system state. It has been used to reason about uncertainty in robotic systems ([Cao93]). However, within a Petri Net, the way membership functions attached to each tokens and certainty factors associated with transitions can be hard to understand. For business users that are involved in constructing the requirement model, a simple modeling convention is much preferred to a sophisticated one. Thus, using Petri Net may not be very practical.

NASA's DDP ("Defect Detection and Prevention) model [Feather02] is a risk management framework designed to aid decision-making during the earlier phases of advanced technology and system development. It utilizes quantitative analysis, which accepts only one numeric value for each required quantitative value. It is unable to represent the actual situations where uncertainty factors exist.

VI. Conclusion and Future Work

The requirement analysis technique proposed in this paper is summarized as follows: first, a model is needed to capture the tradeoffs/synergy between software quality attributes and applicable design alternatives. We have adopted the softgoal framework by *Chung, Nixon, Yu and Mylopoulous* ([Chung99]) in modeling such knowledge for analysis. Second, an inference engine is built to automatically execute the text-encoded softgoal framework. Thirdly, we incorporate *Monte Carlo* simulation to explore the wide range of behaviors in the model, and summarize these behaviors with a treatment-learning tool named TAR2.

From what we have observed in our experiments on both *KWIC* and SR-1 frameworks, our simulation tool successfully discovered consistent behaviors within each framework despite various uncertainty factors, and provided treatment recommendations relevant to business

concerns. As this approach does not require much concrete domain knowledge, time and expenses dedicated to data collection (e.g. appointments with domain experts, gathering surveys) can be minimized. Moreover, TAR2's treatment results pinpointed the most critical decision towards the problem domain, thus users can focus on this key issue and allot less time in discussing the non-critical ones.

The requirement analysis technique presented in this paper is in its preliminary state, and research is in progress. Much remains to be done to investigate the softgoal framework behavior and refine the proposed technique with more real-world case studies. Specifically, work will be done to determine the sensitivity to benefit and cost functions in general softgoal framework. With our connection with NASA IV&V Facility, we are optimistic in receiving more case study materials to conduct further research activities.

VII. Acknowledgement

This research was conducted at University of British Columbia and West Virginia University, partially under NASA contract NCC2-0979. In part, this work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the NASA IV&V Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

VIII. Reference:

- [Abdel-Hamid91] T. Abdel-Hamid and S. Madnick. Software Project Dynamics: An Integrated Approach. Prentice-Hall Software Series, 1991.
- [Akhavi93] M. Akhavi and W. Wilson. Dynamic simulation of software process models. In Proceedings of the 5th Software Engineering Process Group National Meeting (Held at

Costa Mesa, California, April 26 - 29). Software engineering Institute, Carnegie Mellon University, 1993.

- [Bratko01] I. Bratko. Prolog Programming for Artificial Intelligence. (third edition). Addison-Wesley, 2001.
- [Burgess01] Burgess, C.J., Dattani, I. Hughes, G. May, J.H.R., and Rees, K., "Using Influence Diagrams to Aid the Management of Software Change", Requirements Engineering 6(3), pp173-182, 2001.
- [Cao93] T. Cao and A. C. Sanderson: A Fuzzy Petri approach to reasoning about uncertainty in robotic systems, Proc. of IEEE International Conference on Robotics and Automation, p317-322, 1993.
- [Chung99] Y. M. Chung, Nixon. Non-Functional Requirements in Software Engineering. Kluwer Academic Publishers, 1999.
- [Clancey96] 4] W. Clancey, P. Sachs, M. Sierhuis, and R. van Hoof. Brahms: Simulating practice for work systems design. In P. Compton, R. Mizoguchi, H. Motoda, and T. Menzies, editors, Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop. Department of Artificial Intelligence, 1996.
- [Feath02] M. Feather and T. Menzies, Converging on the Optimal Attainment of Requirements, International Conference on Requirements Engineering, 2002.
- [Feather02] Martin S. Feather & Steven L. Cornford. Quantitative Risk-Based Requirements Reasoning; in submission to Requirements Engineering Journal, Model-Based Requirements Engineering, 2002.
- [Harel90] D. Harel. Statemate: A working environment for the development of complex reactive systems. IEEE Transactions on Software Engineering, 16(4):403–414, April 1990.
- [Harell00] H. Harrell, L. Ghosh, and S. Bowden. Simulation Using ProModel. McGraw-Hill, 2000.
- [JPL] NASA Jet Propulsion Laboratory web site: http://www.jpl.nasa.gov

- [Kalos86] M. Kalos and P. Whitlock, Monte Carlo Methods, Volume 1: Basics. New York: J. Wiley, (1986).
- [Kelton02] D. Kelton, R. Sadowski, and D. Sadowski. Simulation with Arena, second edition. McGraw-Hill, 2002.
- [Kohavi95] Kohavi R, A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. IJCAI-95.
- [Law00] A. Law and B. Kelton. Simulation Modeling and Analysis. McGraw Hill, 2000.
- [Martin00] R. Martin and D. M. Raffo. A model of the software development process using both continuous and discrete models. International Journal of Software Process Improvement and Practice, June/July 2000.
- [Me01] T. Menzies; Practical Machine Learning for Software Engineering and Knowledge Engineering; Handbook of Software Engineering and Knowledge Engineering; December; World-Scientific; 981-02-4973-X; 2001.
- [Me02] Tim Menzies, Eliza Chiang, Martin Feather, Ying Hu, James D. Kiper. Condensing uncertainty via incremental treatment learning. Annals of Software Engineering; special issue on Computational Intelligence, 2002.
- [Meh01] Ying Hu, Tim Menzies. Constraining discussions in requirements engineering via models. 2001.
- [Mi90] P. Mi and W. Scacchi. A knowledge-based environment for modeling and simulation software engineering processes. IEEE Transactions on Knowledge and Data Engineering, pages 283–294, September 1990.
- [Nasa] NASA IV&V web site; url: http://www.ivv.nasa.gov/.
- [Parnas72] D. Parnas. On the criteria to be used in decomposing systems into modules. Communications of the ACM, 5(12):1053–1058, 1972.
- [Prolog] SWI-Prolog web site; url: http://www.swi-prolog.org/.
- [Shaw96] M. Shaw and D. Garlan. Software Architecture: Perspectives on an Emerging discipline. Prentice Hall, 1996.

- [Shachter86] Ross D. Shachter. "Evaluating Influence Diagram". Operations Research, Volume 34, Issue 6 Nov. - Dec., 1986, 871-882.
- [Simon57] Simon, H.A. "Rational Choice and the Structure of the Environment," in Simon, H.A. (Ed), Models of Man, John Wiley, NY (1957).
- [Sr102] T. S. Corporation. IV&V Catastrophic/Critical/High Risk Function List for the Demonstration of Autonomous Rendezvous Technology Project, 2002.
- [Sterman00] H. Sterman. Business Dynamics: Systems Thinking and Modeling for a Complex World. Irwin McGraw-Hill, 2000.

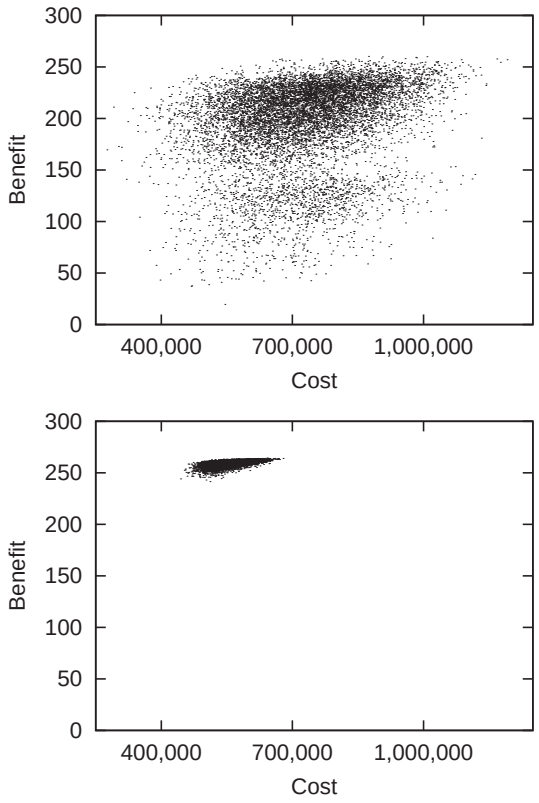

Fig. 1. An example of treatment learning results

Cost	Benefit vvlow	vlow	low	high	vhigh	vvhigh
zero	26	17	10	5	2	1
one	28	19	12	7	4	3
two	30	21	14	9	8	6
three	32	23	16	15	13	11
four	34	25	24	22	20	18
five	36	35	33	31	29	27

TABLE I
CLASS RANKINGS FOR KWIC FRAMEWORK

<combine_logic>	<op>	<arithmetic[op]>
all_of	rand	minimum
any_of	rany	summation
any_one_of	ror	maximum
<contribution>	<value> [contribution]	<arithmetic> [contribution]
helped	mean=1.4	multiply
made	mean=1.8	multiply
unhurt	mean=0.6	multiply
unbroken	mean=0.2	multiply
<priority>	<value> [priority]	<arithmetic> [priority]
veryCritical	mean=2.0	multiply
critical	mean=1.5	multiply
normal	mean=1.0	multiply
<softgoal>	<softgoalType> [softgoal]	<cost> [softgoalType]
operationalizing softgoal	any type	1

Fig. 3. Settings for KWIC framework

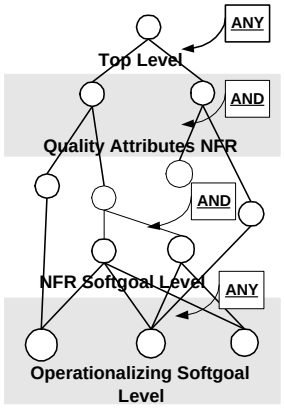

Fig. 4. logic configuration for rigorous quality assurance scheme

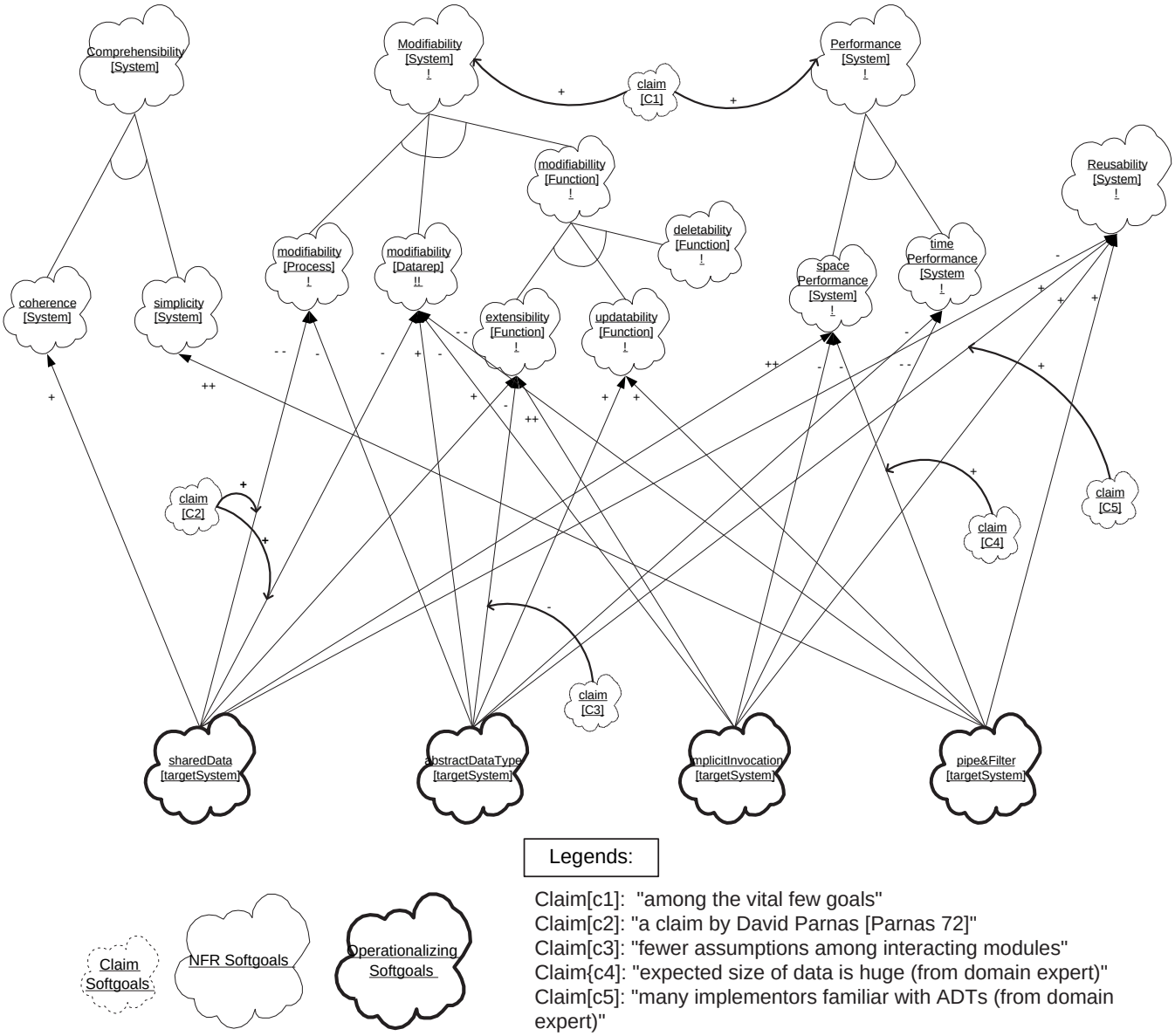

Fig. 2. KWIC framework

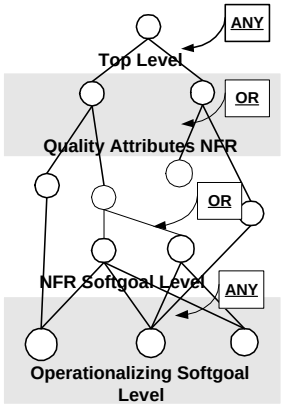

Fig. 5. logic configuration for weak quality assurance scheme

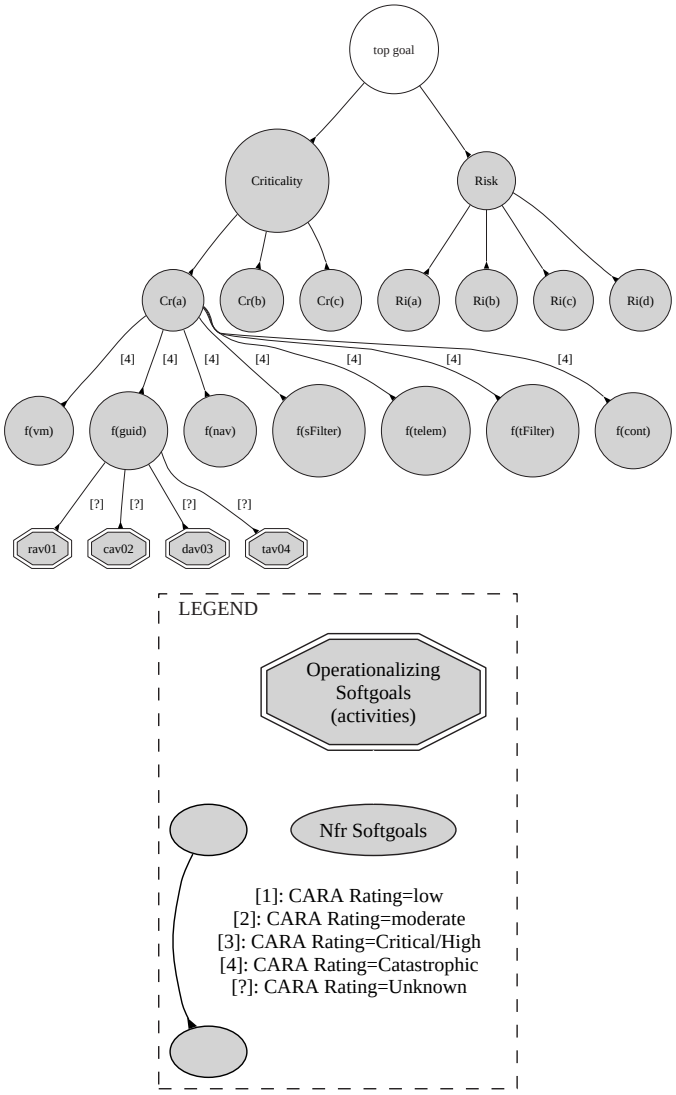

Fig. 6. Segment of the SR-1 framework

Function	Criticality			Risk				Level
	Cr[a]	Cr[b]	Cr[c]	Ri[a]	Ri[b]	Ri[c]	Ri[d]	
f[vm]	4	4	2	3	2	3	3	F
f[guid]	4	4	2	3	2	2	3	F
f[nav]	4	4	2	3	2	2	3	F
f[sFilter]	4	4	2	3	1	2	3	F
f[telem]	4	1	3	3	1	3	2	F
f[tFilter]	4	1	2	3	2	2	3	L
f[cont]	4	4	1	2	1	2	1	L
f[cam]	2	1	1	3	3	2	3	B
f[exInf]	3	2	1	1	1	2	1	N
f[oReqm]	3	3	3	1	2	2	3	F
f[sMode]	3	1	3	3	2	2	3	L
f[aMode]	3	1	3	3	2	2	3	L
f[tMode]	3	1	3	3	2	2	3	L
f[dMode]	3	1	2	3	2	2	3	L
f[sbMode]	1	3	1	1	1	2	2	L
f[pMode]	3	1	1	1	1	2	2	N
f[aexInf]	3	1	1	2	1	2	1	N
f[comm]	3	1	1	1	1	3	1	N

Fig. 7. CARA ratings on SR-1 software functions and corresponding Analysis Level

Cost	Benefit < 5.5	< 11	< 16.5	< 22	< 27.5	< 32	
0							
1	3.86						3.86
2	32.9	0.27					33.17
3	10.01	3.49	0.02				13.52
4	6.66	22.68	5.88	0.45	0.02		35.69
5	1.06	6.74	4.94	0.95	0.07		13.76
total	54.49	33.18	10.84	1.4	0.09		100

TABLE II

ROUND 1 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN IN 10,000 RUNS OF FIG 4; NO TREATMENT

Cost	Benefit < 5.5	< 11	< 16.5	< 22	< 27.5	< 32	
0							
1							
2	27.84						27.84
3	10.71	7.05	0.28				18.04
4	4.26	8.89	1.7	0.11	0.01		14.97
5	2.83	19.2	14.68	2.24	0.19	0.01	39.15
total	45.64	35.14	16.66	2.35	0.2	0.01	100

TABLE III

ROUND 2 CONSTRAINT: $sharedData\ of\ targetSystem = yes$ (RIGOROUS QUALITY ASSURANCE)

Cost	Benefit < 5.5	< 11	< 16.5	< 22	< 27.5	< 32	
0							
1							
2							
3	25.21	2.7					27.91
4	8.86	18.51	2.13	0.04			29.54
5	3.15	21.13	15.5	2.45	0.32		42.55
total	37.22	42.34	17.63	2.49	0.32		100

TABLE IV

ROUND 3 CONSTRAINT: $implicitInvocation\ of\ targetSystem = yes$,$sharedData\ of\ targetSystem = yes$ (RIGOROUS QUALITY ASSURANCE)

Cost	Benefit < 5.5	< 11	< 16.5	< 22	< 27.5	< 32	
0							
1							
2							
3							
4	10.34	24.86	3.52	0.08		‘	38.8
5	4.68	31.01	21.51	3.64	0.34	0.02	61.2
total	15.02	55.87	25.03	3.72	0.34	0.02	100

TABLE V

ROUND 4 CONSTRAINTS: $abstractDataType\ of\ targetSystem = yes$, $c3 = yes$, $implicitInvocation\ of\ targetSystem = yes$,$sharedData\ of\ targetSystem = yes$ (RIGOROUS QUALITY ASSURANCE)

Cost	Benefit < 14.67	< 29.33	< 44	< 58.67	< 73.33	< 87	
0							
1	3.06						3.06
2	6.99	10.36	0.62	0.02			17.99
3	4.18	25.26	11.92	1.41	0.1		42.87
4	1.72	12.88	13.26	3.61	0.31	0.04	31.82
5	0.27	1.57	1.77	0.59	0.06		4.26
total	16.22	50.07	27.57	5.63	0.47	0.04	100

TABLE VI

ROUND 1 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN IN 10,000 RUNS OF FIG 5; (WEAK QUALITY ASSURANCE) NO TREATMENT

Cost	Benefit < 14.67	< 29.33	< 44	< 58.67	< 73.33	< 87	
0							
1	12.34	0.02					12.36
2	9.18	18.66	1.77	0.05			29.66
3	4.71	26.32	16.1	3.24	0.24	0.02	50.63
4	0.5	3.29	2.8	0.7	0.06		7.35
5							
total	26.73	48.29	20.67	3.99	0.3	0.02	100

TABLE VII

ROUND 2 CONSTRAINTS: $c4 = yes$, $pipeAndFilter\ of\ targetSystem = no$ (WEAK QUALITY ASSURANCE)

Cost	Benefit < 14.67	< 29.33	< 44	< 58.67	< 73.33	< 87	
0							
1	11.89	0.01					11.9
2	9.03	18.6	1.95	0.04			29.62
3	4.77	26.17	17.05	3.47	0.29	0.01	51.76
4	0.38	2.77	2.91	0.56	0.1		6.72
5							
total	26.07	47.55	21.91	4.07	0.39	0.01	100

TABLE VIII

ROUND 3 CONSTRAINTS: $c3 = yes$,$c4 = yes$, $pipeAndFilter\ of\ targetSystem = no$ (WEAK QUALITY ASSURANCE)

Cost	Benefit < 14.67	< 29.33	< 44	< 58.67	< 73.33	< 87	
0							
1	20.34	0.05					20.39
2	8.38	28.29	4.62	0.18			41.47
3	1.84	15.66	15.84	4.27	0.48	0.05	38.14
4							
5							
total	30.56	44	20.46	4.45	0.48	0.05	100

TABLE IX

ROUND 4 CONSTRAINTS: $c2 = yes$,$c3 = yes$,$c4 = yes$, $pipeAndFilter\ of\ targetSystem = no$ (WEAK QUALITY ASSURANCE)

Code	Level	Cost
Requirements Analysis Activities		
rav01	B,L,F,C	notHigh
rav02	B,L,F,C	notHigh
rav03	B,L,F,C	notHigh
rav04	B,L,F,C	notHigh
rav05	B,L,F,C	notHigh
rav06	B,L,F,C	notHigh
rav07	B,L,F,C	notHigh
rav08	B,L,F,C	notHigh
rav09	B,L,F,C	notHigh
rav10	F,C	veryHigh
rav11	F,C	veryHigh
rav12	F,C	veryHigh
rav13	F,C	veryHigh
rav14	C	extremelyHigh
Design Analysis Activities		
dav01	L,F,C	high
dav02	L,F,C	high
dav03	L,F,C	high
dav04	L,F,C	high
dav05	L,F,C	high
dav06	L,F,C	high
dav07	L,F,C	high
dav08	L,F,C	high
dav09	L,F,C	high
dav10	F,C	veryHigh
dav11	F,C	veryHigh
dav12	F,C	veryHigh
dav13	F,C	veryHigh
dav14	C	extremelyHigh
Code Analysis Activities		
cav01	L,F,C	high
cav02	L,F,C	high
cav03	L,F,C	high
cav04	L,F,C	high
cav05	L,F,C	high
cav06	L,F,C	high
cav07	F,C	veryHigh
cav08	F,C	veryHigh
cav09	F,C	veryHigh
cav10	F,C	veryHigh
cav11	F,C	veryHigh
cav12	C	extremelyHigh
cav13	C	extremelyHigh
cav14	C	extremelyHigh
Test Analysis Activities		
tav01	B,L,F,C	high
tav02	B,L,F,C	high
tav03	B,L,F,C	high
tav04	B,L,F,C	high
tav05	B,L,F,C	high
tav06	B,L,F,C	high
tav07	B,L,F,C	high
tav08	L,F,C	veryHigh
tav09	L,F,C	veryHigh
tav10	F,C	veryHigh
tav11	F,C	veryHigh
tav12	C	extremelyHigh

Fig. 8. Analysis Activities, applicable Analysis Levels for SR-1's functions, and Cost

Cost	Benefit vlow	low	high	vhigh
vlow	10	5	2	1
low	12	7	4	3
high	14	9	8	6
vhigh	16	15	13	11

TABLE X

CLASS RANKINGS FOR SR-1 FRAMEWORK

Code	Requirements Analysis Activity
Rav01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements.
Rav02	Validate ability of requirements to meet system needs
Rav03	Verify Traceability to and from parent requirements
Rav04	Analyze data/adaptation requirement
Rav05	Analyze Testability, Qualification requirements
Rav06	Analyze Data FnotHigh, Control FnotHigh, moding and sequencing
Rav07	Assess development metrics
Rav08	Analyze development risks/mitigation plans
Rav09	Analyze Timing and Sizing requirements
Rav10	Review developer timing/sizing, loading engineering analysis
Rav11	Perform engineering analysis of key algorithms
Rav12	Review/use developer prototypes or dynamic models
Rav13	Develop alternative static representations (diagrams, tables)
Code	Design Analysis Activity
Dal01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements
Dal02	Validate ability of design to meet system needs
Dal03	Verify Traceability to and from requirements
Dal04	Analyze database design
Dal05	Analyze design Testability, Qualification requirements
Dal06	Analyze design Data FnotHigh, Control FnotHigh, moding, sequencing
Dal07	Analyze control logic, error/exception handling design
Dal08	Assess design development metrics
Dal09	Analyze development risks/mitigation plans
Dal10	Review developer timing/sizing, loading engineering analysis
Dal11	Perform design analysis of select critical algorithms
Dal12	Review/use developer prototypes or dynamic models
Dal13	Develop alternative static representations (diagrams, tables)
Code	Code Aalysis Activity
Cal01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements
Cal02	Verify Traceability to and from design
Cal03	Verify Architectural design compliance (structure, external I/O, and CSCI executive moding, sequencing and control)
Cal04	Verify supportability and maintainability
Cal05	Access code static metrics
Cal06	Verify CSU and CSC level logical structure and control fnotHigh
Cal07	Verify internal data structures and data fnotHigh/usage
Cal08	Verify error and exception handling
Cal09	Verify code and external I/O data consistency
Cal10	Verify correct adaptation data and ability to reconfigure
Cal11	Verify correct operating system and run time libraries
Code	Test Analysis Activity
Tal01	Analyze System level verification requirements to verity that test definition, objectives, plans and acceptance criteria are sufficient to validate system requirements and operational needs associated with CCHR functions
Tal02	Verify Software Test Plan qualification testing methods and plans are sufficient to validate software requirements and operational needs
Tal03	Verify test cases traceability and coverage of software requirements, operational needs and capabilities
Tal04	Verify software STD test case definition inputs, expected results, and evaluation criteria comply with STP plans and testing objectives
Tal05	Analyze correct dispositioning of software test anomalies
Tal06	Validate software test results compliance with test acceptance criteria
Tal07	Verify trace and successful completion of all software test case objectives
Tal08	Verify ability of software test environment plans and designs to meet software testing objectives
Tal09	Verify regression tests are sufficient to determine that the software is not adversely affected by changes
Tal10	Analyze STD procedures for test setup, execution, and data collection; confirm procedures completely and correctly test referenced requirements, confirm inspection and analysis completely verifies referenced requirements
Tal11	Monitor execution of software testing as needed

Fig. 9. Analysis Activities Keys to figure 8

version	1	2
`<notHigh>` (X)	$mean(X) = 1$	$mean(X) = mean(Y) * 0.7$
`<high>` (Y)	$Y = mean(X) * 10$	$mean(Y) = 2$, $0 < Y < 10$
`<veryHigh>` (Z)	$Z = mean(Y) * 10$	$mean(Z) = Y * F$, $mean(F) = 1.2; 1.1 \leq F \leq 1.7$

Fig. 10. Two versions of cost function

`<combine_logic>`	`<op>`	`<arithmetic[op]>`
all_of	`rand`	minimum
any_of	`rany`	summation
any_one_of	`ror`	maximum
`<contribution>`	`<value>` `[contribution]`	`<arithmetic>` `[contribution]`
helped	mean=1.4	multiply
made	mean=1.8	multiply
catastrophically_rated	mean=1.9	multiply
critically_rated	mean=1.4	multiply
highly_rated	mean=1.1	multiply
lowly_rated	mean=0.4	multiply

Fig. 11. Miscellaneous settings for SR-1 framework

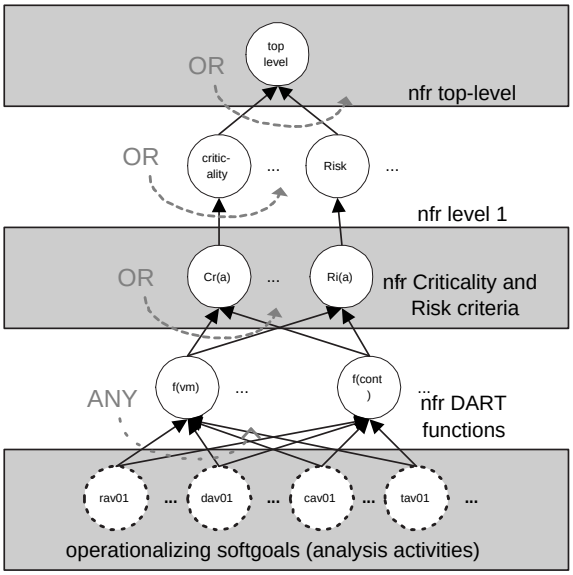

Fig. 12. SR-1 framework: 1

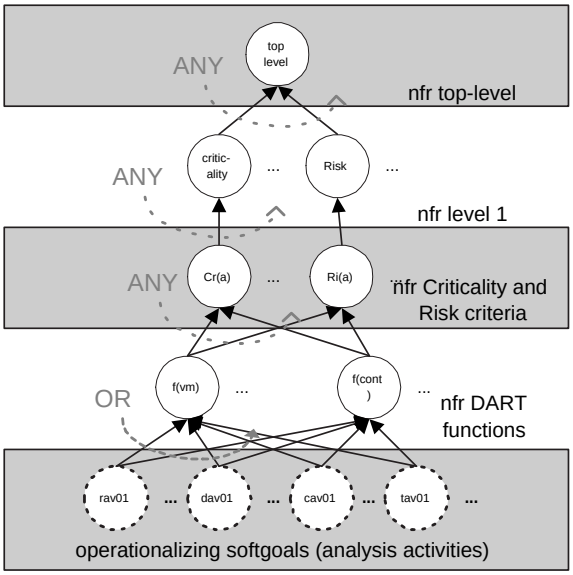

Fig. 13. SR-1 framework: 2

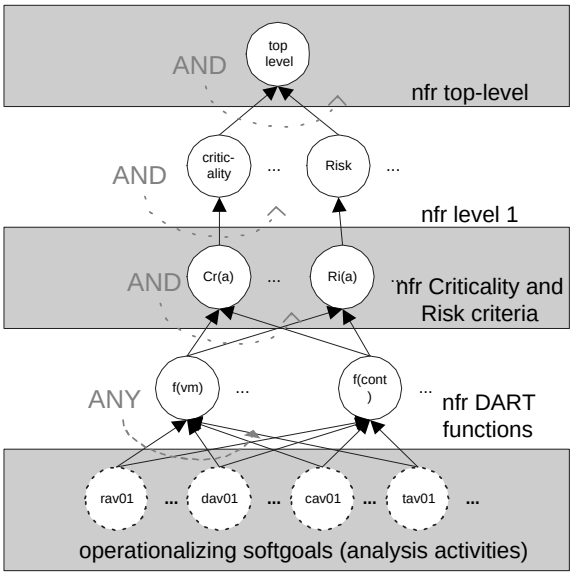

Fig. 14. SR-1 framework: 3

Cost	Benefit vlow	low	high	vhigh	Total
vlow	34.15				34.15
low		4.02	6.26	5.58	15.86
high		6.2	9.98	8.82	25
vhigh		5.64	8.76	10.59	24.99
Total	34.15	15.86	25	24.99	100

TABLE XI

SR-1 FRAMEWORK 1 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN IN 10,000 RUNS OF FIG 12; NO TREATMENT

Cost	Benefit vlow	low	high	vhigh	Total
vlow					
low		4.70	7.74	7.30	19.7
high		9.95	16.0	14.2	40.1
vhigh		9.05	14.1	17.0	40.1
total		23.7	37.8	38.5	100

TABLE XII

MORE PREFERRED SYSTEM: SR-1 FRAMEWORK 1 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN AFTER APPLYING TREATMENTS ($tav09\ of\ tal = y$) FOR A MORE DESIRABLE SYSTEM

Cost	Benefit vlow	low	high	vhigh	Total
vlow					
low					
high		5.16	9.52	8.6	23.27
vhigh		17.32	26.9	32.51	76.73
Total		22.47	36.41	41.11	100

TABLE XIII

LESS PREFERRED SYSTEM: SR-1 FRAMEWORK 1 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN AFTER APPLYING TREATMENTS ($cav10\ of\ cal = y$) FOR A LESS DESIRABLE SYSTEM

Cost	Benefit vlow	low	high	vhigh	Total
vlow	17.63	2.21	2.67	2.5	25.01
low	3.84	8.76	6.16	6.24	25
high	2.48	8	7.12	7.4	25
vhigh	1.06	6.03	9.05	8.85	24.99
Total	25.01	25	25	24.99	100

TABLE XIV

SR-1 FRAMEWORK 2 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN IN 10,000 RUNS OF FIG 13; NO TREATMENT

Cost	Benefit vlow	low	high	vhigh	Total
vlow	25.35	3.13	3.83	3.6	35.91
low	4.88	11.58	8.4	8.66	33.52
high	2.26	7.65	6.67	7.61	24.19
vhigh		1.56	2.16	2.32	6.38
Total	32.84	23.91	21.06	22.18	100

TABLE XV

MORE PREFERRED SYSTEM: SR-1 FRAMEWORK 2 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN AFTER APPLYING TREATMENTS ($dav12\ of\ dal = n$) FOR A MORE DESIRABLE SYSTEM

Cost	Benefit vlow	low	high	vhigh	Total
vlow					
low	3.4	5.69	1.65	2.11	12.86
high	5.51	15.52	7.71	6.98	35.72
vhigh	4.13	12.95	16.71	17.17	50.96
Total	13.22	34.44	26.08	26.26	100

TABLE XVI

LESS PREFERRED SYSTEM: SR-1 FRAMEWORK 2 PERCENTAGE DISTRIBUTIONS OF $benefit$S AND $cost$S SEEN AFTER APPLYING TREATMENTS ($cav07\ of\ cal = y$) FOR A LESS DESIRABLE SYSTEM