

Software Technology Risk Advisor

Gregory A. Toth

USC Center For Software Engineering *
Department of Computer Science
University of Southern California
Los Angeles, CA 90089-0781, USA
gtoth@usc.edu

Northrop Grumman Corporation *
Electronics Systems Division
Hawthorne Site
Hawthorne, CA 90251-5032, USA
g.toth@ieee.org

Abstract

This paper describes the Software Technology Risk Advisor (STRA), a knowledge-based software engineering tool that provides assistance in identifying and managing software technology risks. The STRA contains a knowledgebase of software product and process needs, satisfying capabilities, and capability maturity factors. After a user ranks the importance of relevant needs to his or her project, the STRA automatically infers risk areas by evaluating disparities between project needs and technology maturities. Identified risks are quantitatively prioritized and the user is given risk reduction advice and rationale for each conclusion. This paper presents methods used in the STRA, along with discussions of knowledge acquisition, experimental results, current status, and related work.

Keywords: Knowledge based software engineering, KBSE, software risk management, software technologies, software development tools, expert systems, advisory systems.

1. Introduction

Software risk management is one of several high-payoff areas in the field of software engineering. Many examples of risk management approaches and benefits are found in [1] and [2]. Unfortunately, effective risk management often requires substantial insight into technical, schedule and cost issues and their complex interactions. Such insight is typically found in persons having extensive experience in a given application domain coupled with a broad understanding of software at large. These experts are almost always in short supply, making it difficult to repeatedly apply their expertise across many software projects. One method of capturing expert knowledge

and making it widely available is through knowledge-based tools such as expert systems. Since risk management requires expert knowledge, it is a natural application for knowledge-based approaches.

A knowledge-based tool known as the Software Technology Risk Advisor (STRA) has been developed to provide expert assistance during software risk management activities. The STRA acquires information about a user's project, correlates the information to stored knowledge of software technical issues, and identifies disparities between project needs and current software capabilities. Such disparities are considered a source of risk, primarily due to the need for software technologies that are currently immature or unavailable. Using previously developed or time-proven software technology generally implies lower risk, while needing non-existent or unproven technology generally implies higher risk. Software technology risk has been selected as a subset of the more general technical/schedule/cost risk areas associated with modern software development.

Risk areas are quantitatively determined and prioritized by analyzing multiple project factors using interrelationships stored in the knowledgebase. A prioritized list of risk areas is automatically produced along with their absolute and relative weighting factors. Explanation of each risk area includes a description of the risk and rationale for each inference; this allows fine tuning and provides greater insight into how results were achieved.

Several additional functions are provided as a by-product of basic risk identification. The knowledge sources used as a basis contain much useful information about the current state of software technology, including both products and the processes by which they are created. This knowledge can be useful to users who are not familiar with current software issues. Explanation facilities provide organized access to this knowledge within a structured framework. A broad taxonomy of large-scale software development issues is thereby covered.

The knowledgebase is quantified using opinions of software experts. Less-experienced users can benefit from access to this expertise. Additional benefit is provided by interrelationships and consistency rules derived from ex-

* The author is a software manager at Northrop Grumman and a graduate student at USC.

perts and stored in the knowledgebase.

The STRA also provides a framework for maintaining notes related to individual facets of software technology and engineering. These notes provide insight into various issues and are provided upon user request. As such, the STRA serves as a unified repository for knowledge, relationships, notes, cross-references, and other information pertinent to software technical issues and risk areas.

The value of this STRA tool is to make expert knowledge available to less-expert acquisition managers, program/project managers, software managers, software developers, and other personnel involved with software. Frequently a software technology expert is needed to assess new ideas or proposals. The STRA might allow more frequent assessment by non-expert personnel, especially during concept formation stages. Such assistance could improve the software development process by providing early identification of high-risk approaches, thus allowing more time to develop appropriate risk tradeoffs, risk reduction measures, or risk mitigation techniques.

This paper is organized in sections describing various aspects of the STRA and its development. Section 2 describes the knowledgebase and discusses knowledge acquisition activities. Section 3 describes user interaction with the STRA and internal methods for inferring risks. Section 4 discusses results from experimental use. Section 5 compares the STRA to related work. Section 6 discusses the status of STRA development and future work. These sections are followed by conclusions which summarize salient points.

2. STRA knowledgebase

2.1 Software technology taxonomy

Techniques used in the STRA are not dependent on a specific source of knowledge. However, the U.S. Department of Defense Software Technology Strategy (SWTS) [3] provides a reasonable taxonomy of current software technologies and their relative maturity. This was considered a good starting point for STRA development, and was used as the basis for an initial knowledgebase. Other written knowledge sources, such as briefings, conference proceedings and textbooks, were used as supportive and clarifying material.

The main objective of the SWTS was to define a plan for pursuing specific software technology areas. In order to determine which technologies are most important, it was necessary to identify capabilities needed to support future military systems. The DoD evaluated several long-range military plans to identify system/software needs and software capabilities that satisfy those needs. By qualita-

tively evaluating the maturity of each required capability, the SWTS identifies technologies needing further research and development work.

The SWTS document classifies software needs into two main groups: *Products* and *Process*. Products refer to the end-item software systems being built, while process refers to the techniques used to build them. Each of these groups is further divided into *functions* and *attributes*.

Five categories of software products are identified for DoD military systems: Command, Control, Communications and Intelligence (C³I) systems; Combat Unit systems; Corporate Information Management (CIM) systems; Manufacturing, Science and Engineering applications; and Simulation and Training systems. Each of these areas has different technology needs.

Product functions describe specific capabilities provided by software, such as "*Fuse information from changing, imperfect multiple sources.*" Functions are usually observable at the system level, and are typically elaborated in system or software requirements specifications. Product functions are often domain-specific in nature.

Product attributes are desired *qualities* of a software system, such as "*Interoperability and portability of DoD software applications and reusable software components.*" In general they are less domain-specific than product functions, and while being less tangible, they are often equally as important. Developing software systems possessing desired attributes requires software technologies in much the same way as product functions.

Process functions and attributes, on the other hand, cover technologies used when building or maintaining software. While these may not be as technologically intense as product technologies, they are nevertheless important to successful software development. As software systems continue to grow in size, process technologies become even more important to assuring software quality, timeliness, and cost effectiveness.

Each need is satisfied by one or more *capabilities*, which are also identified in the SWTS. Capabilities are specific applications of technology, such as "*Smart Sensors.*" The SWTS attempts to describe each capability's maturity using simple qualitative statements such as "*Simple aids (adaptive beamforming).*"

2.2 Expert interviews

Domain experts were interviewed to supplement written knowledge sources. Kelly [4] provides many practical guidelines for knowledge engineering. The main thrust of expert interviews is to: 1) Provide *quantitative* values for needs and capabilities; 2) Determine areas of highest payoff; 3) Identify heuristics that relate various

facets of needs, capabilities, and technologies; and 4) Provide feedback on working versions of the STRA. The first part of each expert interview is a discussion of the expert's background in order to establish proper context for questions and answers. A major challenge of STRA development was to identify and mechanize quantitative relationships between knowledge elements; much of this quantification comes from the experts.

Interviewing domain experts is an ongoing activity used to expand and improve the knowledgebase. Two experts were interviewed during initial knowledgebase development. These initial interviews were conducted through a series of meetings which followed a pre-planned agenda. Since the interviewees often had busy schedules, establishing and following a detailed agenda was critical to achieving interview objectives. A concerted effort was made to keep each interview on-track to avoid spending time on low-payoff topics.

The first expert had considerable expertise in software processes and the C³I product domain. The second expert had product experience in the simulation/test and corporate information management domains, including non-DoD and commercial applications. At the end of each interview working STRA prototypes were demonstrated. Results obtained from these interviews included:

- The importance of different product function areas varies based on each person's perspective. For example, someone in the CIM domain may consider CIM systems critically important, while someone in the C³I domain may consider it less important.
- Quantitative values for the relative importance of product and process functions and attributes were obtained; these were used to help formulate default values in the knowledgebase.
- Product functions and attributes tend to vary from project-to-project, while process functions and attributes are more universal.
- Requirements engineering is still generally weak.
- Many laboratory/prototype algorithms and approaches do not scale well to operational systems, particularly in the areas of AI, robustness, and human-computer interfaces.
- Several software technologies frequently deliver less than what is claimed for them. Examples include CASE tools, reusability, COTS software, and Ada.
- Distributed and parallel systems are increasingly needed but are very immature.
- Many observations from past experience were discussed; several of these were converted into rules for

the expert system.

- The general STRA implementation approach, as shown during working demos, seemed reasonable.

2.3 Heuristics, explanations and advice

Heuristics were derived from expert interviews and from the author's own software development experience. Heuristics are used to generate three additional sets of knowledge in the STRA: 1) Consistency rules used to check user responses to questions; 2) Size-effect scale factors used to scale certain risk areas as a function of project size; and 3) Risk reduction advice.

The knowledgebase also contains supporting information for each need, capability, and risk area. This data consists of elaborated descriptions, notes, and cross-references to the knowledge sources. Supporting information is available to the user by selecting appropriate STRA menu commands.

3. Mechanization

3.1 User inputs

Overall user interaction is shown in Figure 1. The STRA tool is typically used in an iterative fashion. After answering questions during the first pass, an initial list of risks is generated. By viewing risk inference rationale, previous answers can be refined in order to improve risk assessment accuracy. Much of the fine tuning takes place by optimizing importance ratings for lower-level capabilities that satisfy each higher-level need. All importance ratings have default values from the knowledgebase; these provide a point of departure and help guide less-knowledgeable users.

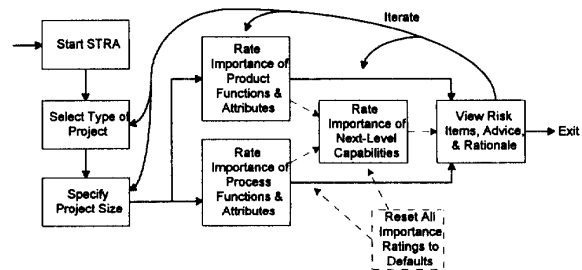


Figure 1: Flow of user interaction with the STRA

3.1.1 Type of project. Each of the five main project types has a different set of needs. Selecting the type of project causes appropriate needs to be displayed so that the user can provide importance ratings for each. A sixth project type is also included; it contains all possible needs

and thus supports projects which do not conveniently fit one of the other categories.

3.1.2 Project size. The maturity of certain software technologies is sensitive to project size. For example, a technology that works well for small projects may not scale well for large projects. Size-effect scale factors are carried in the knowledgebase to handle these conditions. The user specifies project size according to the number of Delivered Source Instructions (defined by Boehm in [5]) as shown in Table 1. Project size selection based on factors other than KDSI (e.g. function points) could be easily accommodated. In general, technologies that are sensitive to project size become less risky as the size decreases.

Project Size	Size Criteria, KDSI
Small	< 10
Medium	10 to 100
Large	100 to 500
Very Large	> 500

Table 1. Project size criteria

3.1.3 Importance of product/process needs. The importance of each relevant product and process need is initially set to a default value derived from experts and stored in the knowledgebase. These defaults correspond to typical projects of each type. Since each user's project may not be completely typical, default importance ratings can be changed to improve risk assessment accuracy. The importance of each need is selected from the set { None, Very-Low, Low, Medium, High, Very-High, Extra-High }.

Each time a rating is set by the user, it is checked against all other ratings by applying consistency rules stored in the knowledgebase. If a conflict or inconsistency is detected, conflicting needs are displayed and the user may re-answer the question or disregard the conflict. Consistency checking provides extra assistance to novice users by immediately pointing out potential problems. It also helps experienced users avoid simple oversights.

3.1.4 Importance of next-level capabilities. Each product or process need is satisfied by one or more capabilities. The importance of each satisfying capability is set to a value in the range { None .. Extra-High }. If the importance of a capability is high, it may have a strong influence on risk results. If the importance is low or none, it may have a weak influence. Capability importance ratings are initially set to defaults from the knowledgebase. The user may wish to change these defaults to better match specific characteristics of the project. For each risk identified by the STRA, detailed rationale shows the quantitative

influence of each needed capability. This is useful for iteratively fine-tuning capability importance ratings as part of optimizing the risk assessment.

3.1.5 Risk description, advice and rationale. After all questions have been answered, the STRA correlates answers and computes numerical risk values for each possible risk. These are sorted in descending order and the top ten risk items are displayed. At this point the user may request detailed information about each risk or repeat the rating/assessment cycle. Detailed risk information includes a risk description, advice on risk reduction, a list of all related technical capabilities and their shortcomings, and all user answers that generated the risk.

An important aspect of the STRA is its explanation capability. The user can select "explain" at most queries to access supplemental information from the knowledgebase.

3.2 Risk Inference

The STRA is implemented as a weighted connection network having four levels as shown in Figure 2. Figure 2 is only representative of the topology; there are actually hundreds of connections. Three of the four levels have weighting factors on each interconnect. A weighted connection scheme allows all factors to be considered simultaneously while retaining a computational (numerical) approach to identifying risks. Key quantitative factors and their use at each connection level is further described below. Each node in the network is implemented by an object-oriented data structure containing all pertinent information along with connections to other nodes. The number of interconnections at each level of the current knowledgebase is summarized in Table 2.

Four primary numerical values, known as *size*, *importance*, *maturity*, and *significance*, were devised to quantify and correlate knowledge. *Size* represents the project size (Table 1) and can take on values from the set { Small, Medium, Large, Very-large } corresponding to numerical values { 3, 2, 1, 0 }.

Importance describes the relative importance of a lower-level need or capability to a higher-level need. *Importance* can take on values from the set { None, Very-Low, Low, Medium, High, Very-High, Extra-High }, corresponding to numeric values { 0, 1, 2, 3, 4, 5, 6 }. Using Figure 2 as an example, *Product Function Needs* has constituent needs *Detect Threats Amid Clutter and Jamming* and *need-48*, where *Detect Threats...* might be of medium importance and *need-48* might be of very-high importance. *Detect Threats...* requires capabilities *Smart Sensors* and *cap-98*, where *Smart Sensors* might be of high importance and *cap-98* might be of extra-high im-

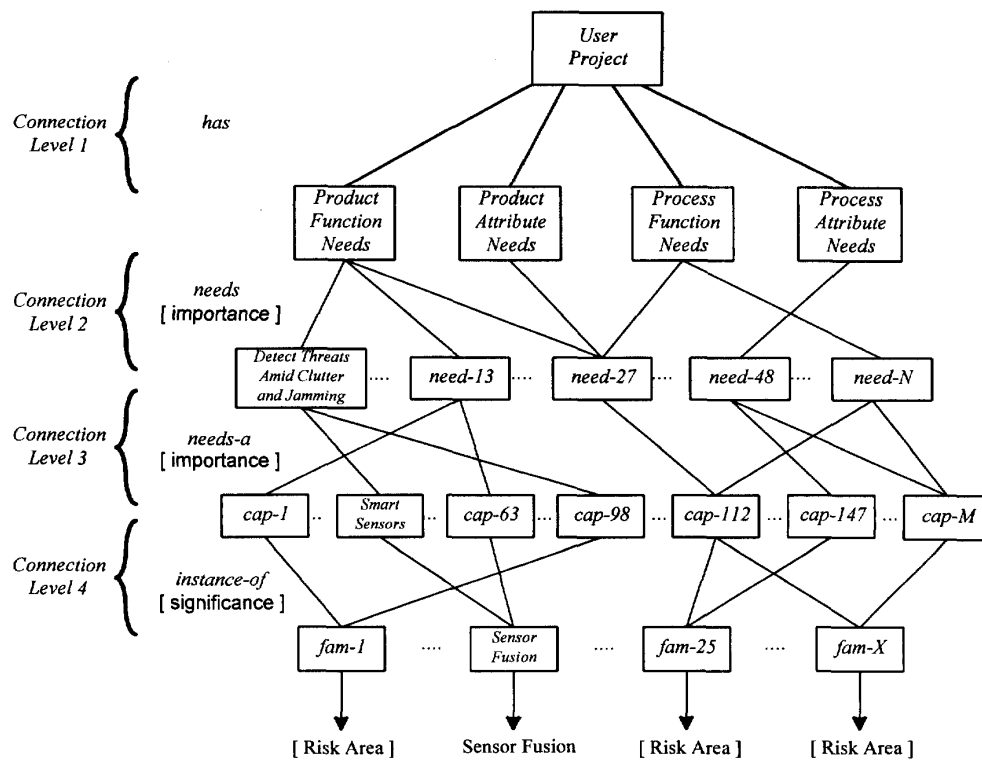


Figure 2: Representative knowledgebase object interconnections

portance. Importance values are used in connection levels 2 and 3 and are specified by the user. Default importance values are obtained from domain experts and are stored in the knowledgebase. Each level 3 connection is also scaled by the *size* factor as described later.

Maturity factors are used solely for capabilities, and identify the relative maturity of a software technology in today's timeframe. *Maturity* can take on values from the same value set as importance, except that { None } is not used. Commonality between values of importance and maturity provides symmetry and consistency in the knowledge representation. It also allows reuse of several data manipulation functions in the implementation. All maturity factors have constant values obtained from domain experts.

Significance represents the relative significance of each mismatch between needs and capabilities. It is computed as the difference between a level 3 importance and a capability's maturity factor, multiplied by the corresponding level 2 importance; the result is then scaled by the size factor. A *size-effect scale factor* is associated with certain capabilities (those which are sensitive to size) to represent the net scale factor for a small project. Linear interpolation is used to compute scale factors for other

sizes. Using Figure 2 as an example, a sample significance calculation is shown below:

Assume:

Project size is large { 1 }

Detect Threats Amid Clutter and Jamming importance to Product Function Needs is very-high { 5 }

Smart Sensors importance to Detect Threats Amid Clutter and Jamming is high { 4 }

Smart Sensors maturity is low { 2 }

Smart Sensors size-effect scale factor is 0.4 (net scale factor for project size = small)

Then:

Smart Sensors significance = $(4 - 2) * 5 = 10$

Applying the *size-effect scale factor* yields:

Smart Sensors significance = $10 * [1.0 - (1 * ((1.0 - 0.4) / 3))] = 8$

Significance is computed with a floor of zero (0) in order to prevent negative values. The *size-effect scale factor* accounts for changes in a capability's significance

Project/Product Type	Level 2 Connections	Level 3 Connections	Level 4 Connections	Product Function Needs	Product Attribute Needs	Process Function Needs	Process Attribute Needs	Software Capabilities	Conflict Checking Rules
C ³ I	28	135	28	7	10	7	4	113	61
Combat Unit	30	129	29	9	10	7	4	110	44
CIM	22	125	26	1	10	7	4	107	33
Mfg. Science & Engrg	24	118	27	3	10	7	4	99	30
Simulation & Training	23	122	26	2	10	7	4	98	30
User-Defined	43	185	33	22	10	7	4	153	78

Table 2. Current knowledgebase metrics

as a function of product size. If the size-effect scale factor is one (1) then no scaling takes place and significance is considered independent of size.

After computing each capability's significance, all capabilities are coalesced according to their connections to capability families. A *capability family* is a general technology area (e.g. real-time systems), and all capabilities belong to at least one family. The significance values of all capabilities in a family are averaged and the result becomes that family's absolute risk weight. Families are sorted in order of decreasing risk weight and the top ten items are presented as potential risk areas. The weightings of each family are also normalized to the highest risk weighting; this makes it easier to see relative risk positioning.

The STRA architecture consists of schemata, rules, and utility functions. Schemata, also known as frames, store basic knowledge about needs and capabilities. Schemata are interrelated through inheritance and through relations such as *has*, *needs*, *needs-a*, and *instance-of*. Importance and size-effect factors are carried on non-inheritance relations to quantify each relationship. Attached to each schema are textual descriptions taken from the SWTS document, an elaborated description, applicable notes, cross-reference information, and advice. All user input, data manipulation, risk computations, consistency checks, and resulting printouts are performed by supporting functions.

3.3 Consistency rules

Consistency rules from the knowledgebase are used to detect potential conflicts between level 2 importance ratings. Each rule consists of three ordered elements in the form (*left*, *right*, *threshold*). When a need corresponding to *right* is rated by the user, it is checked against all active rules to determine which conflicts may exist. Active rules are those which are not being disregarded or ignored. A conflict is asserted if the difference between

the new rating for *right* and the existing rating for *left* is greater than *threshold*, either positive or negative. Any rule involving a *left* need still having a default rating is temporarily ignored in order to prevent needless conflict assertions during initial ratings. Detected conflicts are displayed along with related *left* ratings as illustrated in Figure 4. Table 2 shows the number of consistency rules in the current knowledgebase.

4. Experimental results

4.1 Sample screens

Sample STRA screens are shown in Figures 3 through 5. Figure 3 shows the initial screen after startup, with a Very-Large C³I project as the default. From here the user would select the appropriate project type/size and then rate importance factors for relevant needs. The set of needs changes, based on the knowledgebase taxonomy, whenever the project type is changed. A scrollbar is used to expose needs that are not currently visible due to screen limitations.

All importance ratings have default values supplied by the knowledgebase; these can be changed by making corresponding checkbox selections. A special symbol next to each need shows whether it has been rated or is still at the default. A button is provided to reset all importance ratings to default values; this is useful for returning to a known state.

Each time an importance rating is set, consistency rules are consulted to check for conflicts with other ratings. If a conflict is detected, a dialog box is displayed such as shown in Figure 4. At this point the user has several options for dispositioning the conflict. If a conflict is disregarded, the associated consistency rules will never be checked again. The total number of disregarded conflicts is shown by a gauge in the upper right corner. Controls are provided to turn conflict checking on or off and to reinstate any disregarded conflicts.

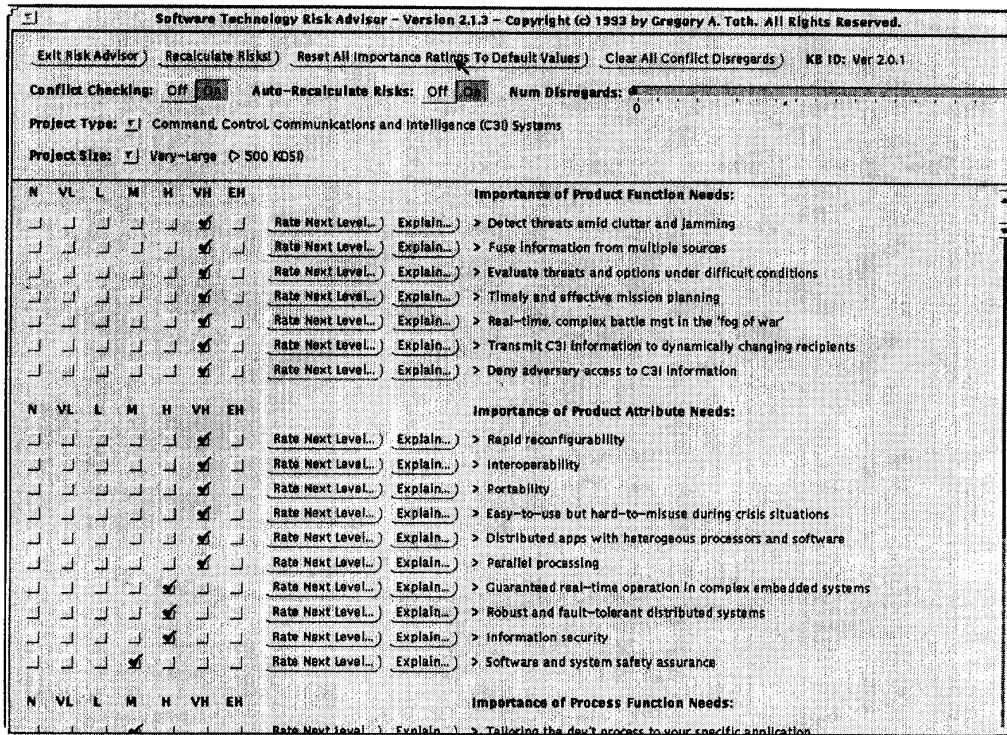


Figure 3: Initial STRA screen showing Level 1 and Level 2 needs

Explanations are available by pressing *Explain* buttons. Lower-level capability ratings are accessed by pressing *Rate Next Level...* buttons. At any time a risk assessment can be performed by pressing the *Recalculate Risks!* button. This causes a list of risks to be displayed as shown in Figure 5. A scrollbar can be used to access additional risks not currently visible. When automatic risk recalculation is turned on, risks are automatically recalculated and displayed each time an importance rating is changed.

Risks are shown in priority order along with their absolute (second column) and normalized (third column) weighting factors. Explanation, advice and rationale for each risk are accessed by selecting the risk as shown in Figure 5. The rationale section shows all capabilities contributing to the risk, along with their significance and maturity factors, maturity descriptions, and originating needs. Pop-up windows can be pinned so they can be viewed while fine-tuning importance ratings.

4.2 Usage results

As of this writing, most STRA usage has been under relatively uncontrolled conditions and during demonstrations. The STRA has been applied to several sample proj-

ects, primarily in the C³I and Combat Unit domains. During experimental use, an initial risk assessment was usually generated after spending five to ten minutes rating level 2 needs. In most cases the initial risk assessment was close to expectations but with perhaps one or two risks which did not seem intuitive. Examination of rationale showed that certain capabilities assumed to be needed (by virtue of their default level 3 importance ratings) were really not applicable for the given project. Once those ratings were optimized through several iterative passes, the assessed risks were usually quite intuitive. In several cases the STRA also identified risks previously unknown to the user; examination of rationale confirmed that they were indeed concerns to the project.

Some of the most useful assistance came from consistency checking rules. On many occasions users realized important but subtle relationships after the STRA identified rating conflicts. For example, if *Transmit C3I Information To Dynamically Changing Recipients* is highly important, *Information Security* and *Process Support For Defect-Free Software Development and Modification* should also be highly important due to the need to prevent compromise of information. Such assistance is invaluable to less-expert users, even if only to stimulate deeper thought.

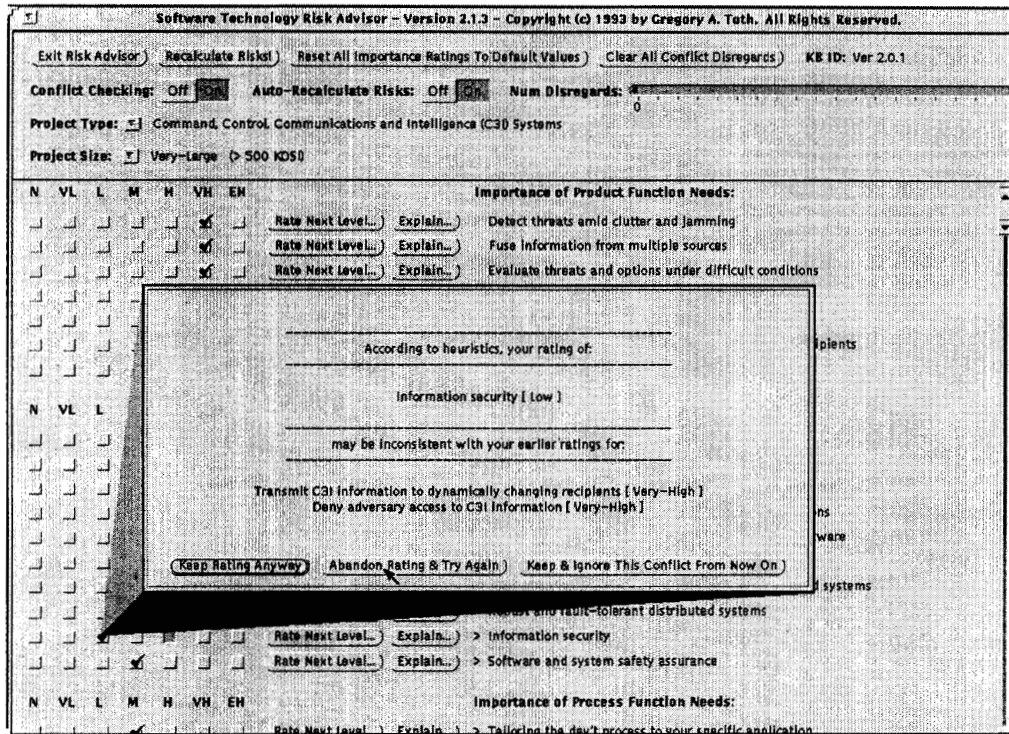


Figure 4: Conflict detected after rating importance of *Information Security*

Several shortcomings were identified as experimental usage progressed; most are discussed in section 6.1. Two additional shortcomings were also noticed:

- The SWTS taxonomy and assessments are geared for large-scale software projects. STRA results for small projects are sometimes erroneous due to insufficient knowledge about scaling effects. This matter will be corrected through further knowledge acquisition.
- Under certain conditions the averaging function used to compute capability family significance is inadequate. A family having a large number of low-significance capabilities can mask a few high-significance capabilities and produce a lower absolute weighting. An alternative is to discard zero-significance capabilities, but this causes a problem for importance values of { None }. Other statistical methods are being investigated.

5. Related work

5.1 Expert COCOMO

Madachy [6] has extended Mitre's work on ESCOMO [7] by developing a knowledge-based assistant

for software cost estimation and project risk assessment. In addition to computing intermediate COCOMO results from user inputs, Madachy's tool identifies risks, detects user input anomalies, and provides advice.

Risks are identified by detecting combinations of upper extreme cost drivers such as *tight schedule and a highly complex system*. Cost driver combinations are evaluated using rules from a knowledgebase. Each risk combination has an associated rating such as moderate, high, or very-high. Identified risks are tabulated according to their membership in five different categories. Risk weights are computed as a function of risk probability and risk consequence.

Consistency rules are used to detect potential user input anomalies such as *embedded mode and very-low complexity*. Advice is provided by displaying information about each risk from the knowledgebase.

Madachy's tool uses simpler inferencing and computational methods than the STRA. Its knowledgebase is smaller and thus provides relatively simple advice and explanations.

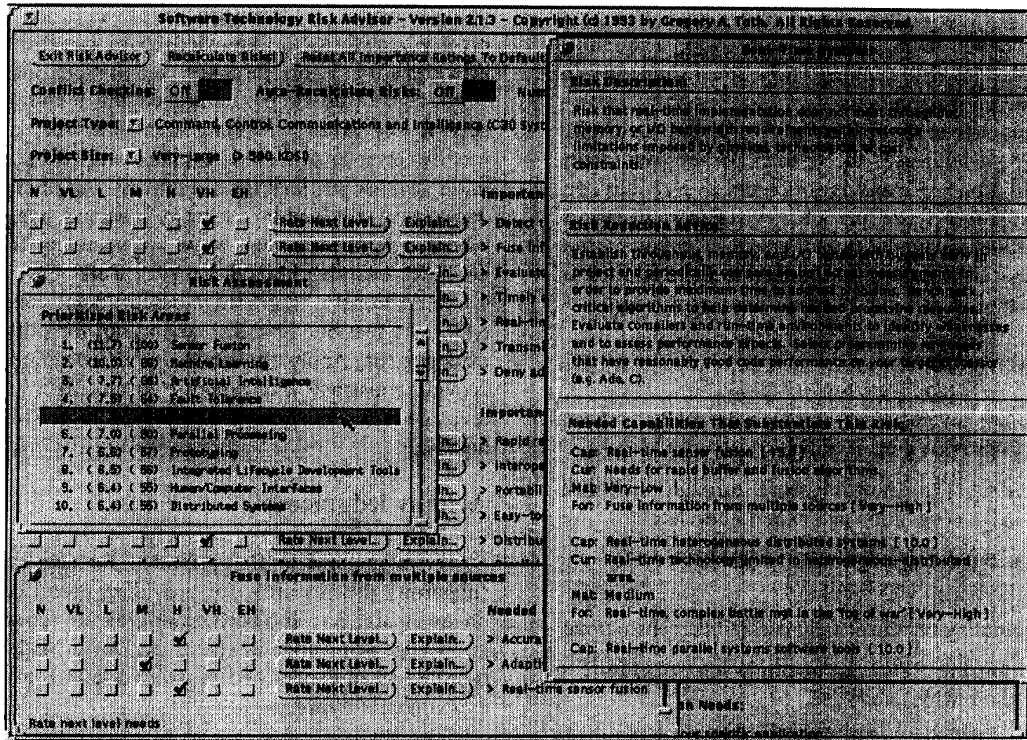


Figure 5: Example risk assessment, Level 3 needs, advice and rationale for *Real-Time Systems*

6. Status and future work

6.1 Implementation versions

STRA development has progressed through several implementation versions, each building on the previous. Version 1 was implemented in Lisp using the KnowledgeCraft expert system shell [8][9][10], which provided a schema-based knowledge representation framework running on a Sun SPARC platform. The Version 1 user interface used simple TTY-oriented menus and queries and provided simple risk information displays. Version 1 successfully demonstrated the risk inference algorithms, but had several shortcomings:

- User interface was cumbersome and forced a sequential question/answer dialogue; this did not allow easy back and forth cycling between user inputs and inferred results.
- Development and execution required Knowledge-Craft expert system shell installed and licensed on the host platform; this limited the number of machines that could support the STRA.
- Knowledgebase was implemented in the Lisp code, making knowledge maintenance difficult for non-

Lisp programmers.

- Simple TTY interface did not readily permit user modification of Level 3 capability importance values; this is needed to allow fine-tuning of the project profile.

Version 2 was written entirely in C and C++ using the Xview toolkit [11] running under Sun OpenWindows™. It provides a graphical user interface (shown in Figure 3) which is much easier to use and allows random user inputs. All knowledge representations and manipulations as well as rule evaluations are implemented in C/C++. Since Lisp is not used, the KnowledgeCraft environment is not required and the STRA can operate on any Sun SPARC platform equipped with a standard windowing environment. This makes it much more accessible to the software community at large.

The Version 2 design decouples the user interface and risk computation engine from the knowledgebase, allowing each to be maintained separately. A knowledgebase compiler was written to transform textual knowledgebase files into binary representations used at runtime; this allows end users to modify supplied knowledgebases or develop their own. Version 1 shortcomings were eliminated, however several desired features were identi-

fied during experimental use. These included printing hardcopy results, saving and restoring importance ratings between sessions, and running on Personal Computers in addition to Sun workstations.

Version 3 is being written in C++ under Microsoft® Windows™. It addresses the desired features and shortcomings of Version 2 and is structured as a toolkit containing the Risk Advisor and a Knowledgebase Compiler. The widespread availability of Personal Computers running Microsoft Windows should lead to increased use of the STRA toolset.

6.2 Future work

There are several areas in which the STRA is being enhanced to provide greater utility:

- Expanding the knowledgebase through additional knowledge acquisition and expert interviews.
- Enhancing knowledge about small projects and commercial systems domains.
- Incorporating results of field use and demonstrations.
- Providing the STRA on multiple host platforms to allow more widespread use.

Other areas are being researched to see how the STRA can be extended and integrated into the software engineering process:

- Adding cost and schedule effects to risk identification and prioritization.
- Providing knowledge-based assistance for deciding which risks to accept, perhaps by looking at risk exposure or economic impact.
- Determining how the STRA best fits into risk management practices and software engineering environments.
- Looking for other applications of the STRA such as assessing technical risks during software design, implementation, and integration/test.

7. Conclusion

This paper has described the Software Technology Risk Advisor, including its internal architecture and knowledgebase structure. The STRA has provided useful technical risk advice on almost all of the projects it has been applied to. It provides low-cost, quick-turnaround assistance for risk identification, risk prioritization, and risk reduction.

The DoD SWTS taxonomy and technology assessments have provided a viable starting point for the STRA

knowledgebase. A taxonomy-based approach supports orderly growth as the knowledgebase is expanded, and provides a framework for notes, explanations and advice.

Usage feedback has improved the initial STRA in several key areas, including user interface, ability to modify lower-level factors, explanations, and support for multiple host platforms/environments. Additional usage and testing will lead to further refinements. Ongoing research may identify new ways of expanding STRA capabilities and tying into other related work.

In summary, knowledge-based tools such as the STRA are feasible, practical, and useful for software risk management. Applying knowledge-based approaches to provide assistance and expertise is an effective way to leverage risk management techniques in software engineering.

Acknowledgments

I am grateful to Barry Boehm for suggesting and supporting this research, for providing useful comments on my work, and for participating as an expert during knowledge acquisition activities. K. David Neal also participated as an expert, and provided useful comments on evolving prototypes.

References

- [1] Boehm, B. W., *Software Risk Management*, IEEE Computer Society Press, Washington, D.C., 1989
- [2] Charette, R. N., *Software Engineering Risk Analysis and Management*, Intertext Publications/Multiscience Press and McGraw-Hill, New York, NY 1989
- [3] United States Department of Defense, "Draft Department of Defense Software Technology Strategy," December 1991
- [4] Kelly, R., *Practical Knowledge Engineering*, Digital Press, 1991
- [5] Boehm, B. W., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981
- [6] Madachy, R., "Knowledge-Based Assistance for Software Cost Estimation and Project Risk Assessment," *Proceedings, Eighth International Forum on COCOMO and Software Cost Estimation*, Software Engineering Institute, October 1993.
- [7] Day, V. C., "Expert System Cost Model (ESCOMO) Prototype," *Proceedings, Third Annual COCOMO User's Group Meeting*, Software Engineering Institute, November, 1987
- [8] Carnegie Group, Inc., "CRL-OPS," June 1990
- [9] Carnegie Group, Inc., "CRL Technical Manual," June 1990
- [10] Carnegie Group, Inc., "Knowledge Craft - Technical Overview," 1990
- [11] Heller, D., *XView Programming Manual*, O'Reilly & Associates, Sebastopol, CA, 1991