[10] Gilb, T., *Principles of Software Engineering Management*, Addison-Wesley Pub. Co., 1988.

[11] Kazman, R., Bass, L., Abowd, G., and Webb, M., “SAAM: A Method for Analyzing the Properties of Software Architectures,” *Proceedings of the 16th International Conference on Software Engineering (ICSE-16)*, IEEE Computer Society Press, Sorrento, Italy, May, 1994, pp. 81-90.

[12] Kazman, R. and Bass, L. “Toward Deriving Software Architectures From Quality Attributes,” CMU/SEI-94-TR-10, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, August, 1994.

[13] McCall, J. and Richards, P., and Walters, G., “Factors in Software Quality,” General Electric Command & Information Systems Technical Report 77CIS02 to Rome Air Development Center, Sunnyvale, CA, 1977.

[14] Robinson, W.N., and Fickas, S., “Automated Support for Requirements Negotiation”, AAAI-94 Workshop on Models of Conflict Management in Cooperative Problem Solving, 1994, pp. 90-96.

References

[1] Basili, V. and Rombach, H., "Tailoring the software process to project goals and environments," *Proceedings of the 9th International Conference on Software Engineering (ICSE-9)*, IEEE Computer Society Press, March, 1987, pp. 345-357.

[2] Boehm, B. and Ross, R., "Theory W Software Project Management: Principles and Examples," *IEEE Trans. on Software Engr.*, July 1989, pp. 902-916.

[3] Boehm, B., Brown, J., Kaspar, H., Lipow, M., MacLeod, G., and Merritt, M. *Characteristics of Software Quality,* TRW Series of Software Technology Report TRW-SS-73-09, TRW Systems and Energy, Inc., 1973.

[4] Boehm, B., Bose, P., Horowitz, E., and Lee, M., "Software Requirements As Negotiated Win Conditions," *Proceedings of the First International Conference on Requirements Engineering (ICRE94)*, IEEE Computer Society Press, Colorado Springs Colorado, April, 1994, pp. 74-83.

[5] Boehm, B., Bose, P., Horowitz, E., and Lee, M.,"Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach," *Proceedings of the 17th International Conference on Software Engineering (ICSE-17)*, IEEE Computer Society Press, Seattle, April, 1995.

[6] Boehm, B., "Some Steps Toward Formal and Automated Aids to Software Requirements Analysis and Design," *Proceedings of IFIP 74*, August, 1974, pp. 192-197.

[7] Chung, L., Nixon, B., and Yu, E., "Using Non-Functional Requirements to Systematically Support Change," *Proceedings of the Second International Conference on Requirements Engineering*, IEEE Computer Society Press, March, 1995.

[8] Easterbrook, S., "Handling conflict between domain descriptions with computer-supported negotiation", *Knowledge Acquisition*, March, 1991, pp. 255-289

[9] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Pub. Co., 1995.

5. Conclusions

The primary conclusions from the initial QARCC-1 experiment were:

- QARCC-1 can help users, developers, customers, and other stakeholders to identify conflicts among their software quality requirements, and to identify additional potentially important quality requirements. In the experiment, QARCC-1 found both of the quality conflicts previously identified, plus five others considered significant.
- QARCC-1 needs further refinement to avoid overloading users with insignificant quality conflict suggestions. In the experiment, QARCC-1 suggested three potential conflicts considered insignificant. We are refining the knowledge base to address more detailed quality attributes in a more selective fashion.

Two additional conclusions derive from feedback based on demonstrations of QARCC-1 to a number of USC-CSE's industry and government Affiliates. There was a strong consensus that QARCC-1 provided a useful framework for stakeholders to systematically resolve software quality attribute conflicts, and that the semi-automated approach provided a good way to balance human skills and computer tools in addressing quality tradeoff issues.

Two additional conclusions are more tentative, awaiting the development of and larger-scale experimentation with QARCC-2. These are that the QARCC approach is usefully scalable to large systems with many quality conflicts, and that the effectiveness of the QARCC approach is largely domain-independent.

Acknowledgments:

This research is sponsored by the Advanced Research Projects Agency (ARPA) through Rome Laboratory under contract F30602-94-C-0195 and by the Affiliates of the USC Center for Software Engineering. The current Affiliates are Aerospace Corporation, Air Force Cost Analysis Agency, AT&T, Bellcore, DISA, Electronic Data Systems Corporation, E-Systems, Hughes Aircraft Company, Interactive Development Environments, Institute for Defense Analysis, Jet Propulsion Laboratory, Litton Data Systems, Lockheed Martin Corporation, Loral Federal Systems, Motorola Inc., Northrop Grumman Corporation, Rational Software Corporation, Rockwell International, Science Applicatons International Corporation, Software Engineering Institute (CMU), Software Productivity Consortium, Sun Microsystems, Inc., Texas Instruments, TRW, U.S. Air Force Rome Laboratory, U.S. Army Research Laboratory, and Xerox Corporation.

ware architectures and quality attributes, but leave open the question of sufficiency of the representative operations. Kazman and Bass [12] explores the relationship between architectural “unit operations”, and a method for deriving software architectures from quality attribute requirements, using a set of eight quality attributes. They provide a useful first-order conflict analysis of the interaction between the eight quality attributes, which we have used and extended in our analyses, but the method of deriving architectures from requirements is somewhat oversimplified. Our assessment is that finding the right balance among conflicting quality attributes is too complex for simple algorithms, and that providing options and suggestions for stakeholders and architects is likely to have more payoff at this time.

The design patterns in Gamma et. al. [9] provide additional candidate attribute strategies, particularly in the areas of Evolvability/Portability, Interoperability, and Reusability. We are analyzing these to capture extensions to the QARCC knowledge base. Robinson and Fickas [14] provide a model and a tool (called “Oz”) for 1) automated conflict detection and resolution, 2) an interactive resolution choice procedure, and 3) records of the negotiation process like our WinWin and QARCC. Their approach requires a domain-dependent knowledge base covering very detailed-level conflicts (e.g., conflicts of loan period in a library system requirements). In contrast our approach focuses on domain-independent conflicts involving high-level quality-attribute and architecture-strategy conflicts in order to achieve generality and scalability.

other situations as well. If so, we plan to drop them as being more time-consuming than beneficial.

	Not Found	Found, Significant	Found, Insignificant
Surfaced in WinWin Exercise	0	2	0
Not Surfaced in WinWin Exercise	0	5	3

Table 3. Conflicts Identified by QARCC-1

4. Related Work

An initial approach to specifying quality attributes was the Requirements-Properties Matrix in [6]. It provided a cross-impact matrix between the software functional requirements and the required properties or attributes, which served as a manual framework for identifying derived functional requirements implied by the attribute requirements. Several of the Rome Laboratory Quality Metrics reports, such as [13], provided checklists of attribute capabilities to be considered in requirements specifications, but did not address automated conflict analysis.

Gilb [10] provides a framework for finding and specifying desired attribute levels in terms of solution specification, tagging, hierarchies, modularization, and cross-referencing, but no resolution aids are provided. Easterbrook [8] provides a good conceptual framework for conflict resolution between domain descriptions with computer-supported negotiation. He provides an approach for resolving conflicts between different domain specifications, and provides an example using a specification of a library system. Chung et. al. [7] provides a good system framework for increasing traceability of quality attributes when changes of quality attribute, their importance, or design decisions and rationale happen during the development process. It draws on domain knowledge to aid in assessing quality attributes, whereas our approach is domain-independent (although tailorable to specific domains).

Kazman et. al. [11], provide a five-step method for analyzing software architectures by analyzing three separate user interface architectures with respect to the quality of modifiability. They use representative operations to analyze the relationship between soft-

- Interoperable SEE functions and tools
- Low development risk
- Low maintenance cost; easy to modify
- Commercializable middleware and commercially supported SEE to improve Evolvability
- Broadly applicable across product line to improve Evolvability

The main objective of the WinWin exercise was to determine the ability of WinWin to support renegotiation of a new win-win equilibrium solution once a new Win Condition was added to the base of 21 in-equilibrium Win Conditions. The new Win Condition, "Support the development of multi-mission satellite ground stations," caused a cost and schedule conflict with the previous negotiated equilibrium. After determining that WinWin could successfully support such a renegotiation[5], we decided to apply QARCC-1 to the body of Win Conditions to see how many potential conflicts it would identify.

First, we wanted to see if QARCC would identify the two conflicts used in the renegotiation process. These were conflicts of Cost/Schedule with Evolvability and Interoperability, which were used by the stakeholders to reject an option to recover cost and schedule by reusing some legacy software which was deficient in Evolvability and Interoperability. Second, we wanted to see if QARCC would surface other potential conflicts, and if so, how many of them would have significant relevance to the satellite ground station system.

The results are shown in Table 3 below. QARCC-1 did find the two significant conflicts surfaced in the WinWin exercise. In addition, if found eight more potential conflicts. Five of these were considered significant in the satellite ground station situation: conflicts of Cost/Schedule with Assurance, Performance, and Reusability; and conflicts of Interoperability and Evolvability with Performance. Three further potential conflicts were not considered significant: conflicts of Evolvability with Assurance and Usability, and a conflict of Cost/Schedule with Usability. We are reviewing these three "false alarm" situations to determine if the potential-conflict threshold for them was set too low for most

By clicking the *Options* button at the bottom of Screen 3, the stakeholder can have QARCC draft a set of candidate resolution Options (left window in Screen 5). QARCC-1 suggests six Options from its knowledge base: 1) “Reduce or defer product functions”; 2) “Find, incorporate some relevant reusable software”; 3) “Find, engage expert performers”; 4) “Use design-to-cost process; Identify lower priority features to defer if necessary”; 5) “Relax constraints on schedule, performance, hardware, and other attributes”; and 6) “Use better tools and practices” to resolve the conflict between Cost/Schedule and Portability. As the Options are generalized, stakeholders can tailor them to their special situations. QARCC-1 also drafts pros and cons for the Options (right window in Screen 5), helping the stakeholders to evaluate the Options and to converge on a mutually satisfactory (i.e., win-win) Option, which is adopted by an Agreement.

3.2 Experimental Results

QARCC-1 has been applied to several sample projects, primarily in the Satellite Ground Stations area. The experiment described here involved applying QARCC-1 retroactively to the Win Conditions for a representative Software Engineering Environment (SEE) to support a Satellite Ground Station (SGS) product line.

The representative Developer was a workstation vendor’s CASE division, the representative User was a large Aerospace Ground Systems Division, and the representative Customer was the hypothetical U.S. Space Systems and Operations Command.

The multi-stakeholder WinWin exercise generated 21 Win Conditions, including the following:

- SGS domain extensions to general SEE (SEE-GEN)
- SGS workload modeling, information flow simulation (SIMU)
- SGS-domain test simulators, test/debug tools (TEST)
- SGS workstation usage scenario generation (USAGE)
- SGS-domain data reduction and reporting (DR&R)
- Initial operational capability (IOC) cost below $7M
- Full IOC delivery schedule within 25 months

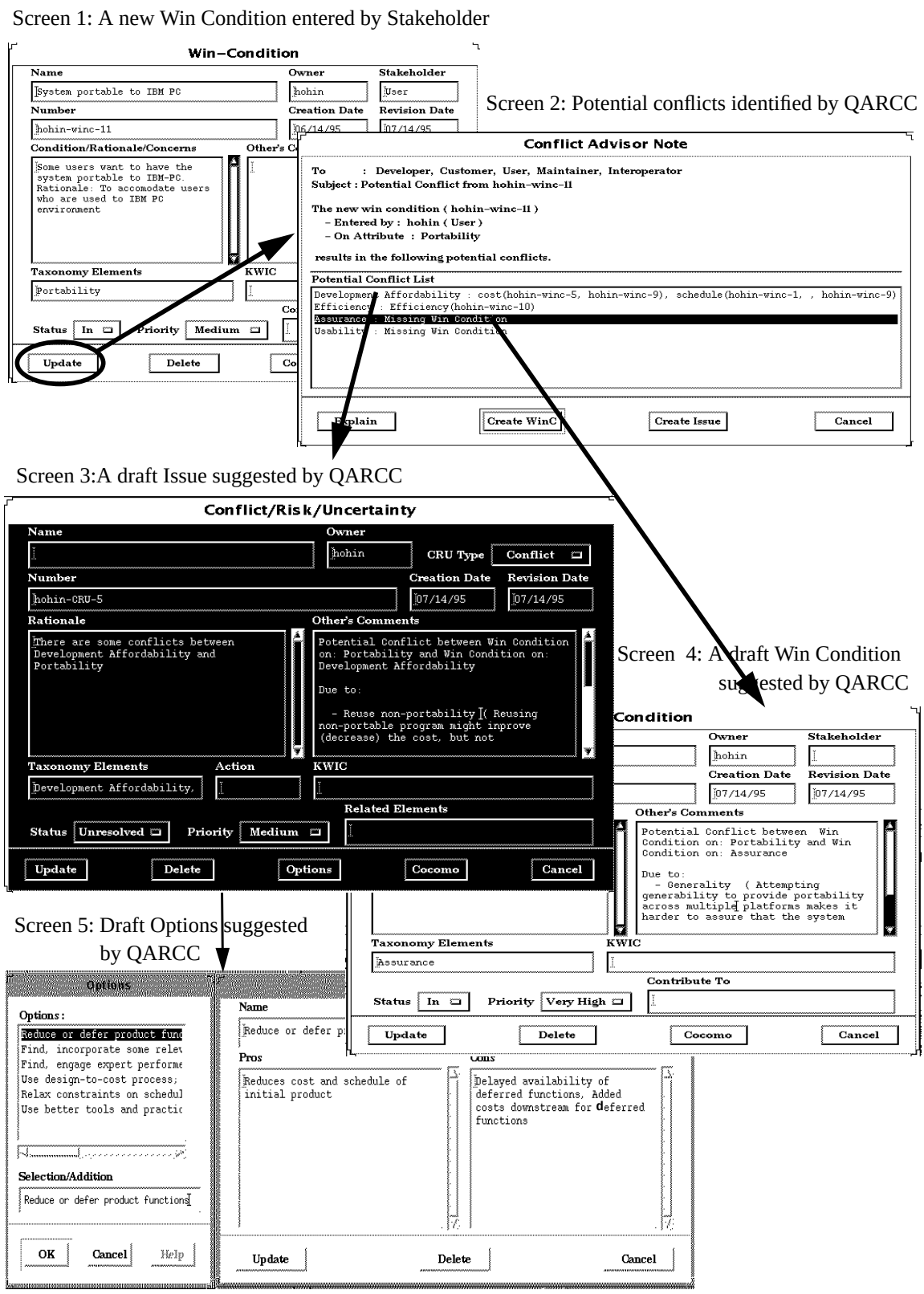


Figure 9. An Example of the Initial Implementation of QARCC

Figure 9 (In the initial implementation, QARCC-1, Cost and Schedule were combined into Development Affordability, and Performance was called Efficiency).

QARCC-1 then uses the relationships in Figure 5 to identify the stakeholders affected by these potential conflicts (Developer and Customer for Cost & Schedule; User and Customer for Performance). For these stakeholders, QARCC-1 pops up the *Conflict Advisor Note* window (Screen 2) which shows a message to directly-concerned stakeholders indicating a list of potential conflicts with the new Win Condition in list form (Potential Conflict List shown in Screen2). The list also enumerates the existing Win Conditions having one of the conflicted attributes in its Taxonomy Elements slot. If there are no existing Win Conditions for a conflicted attribute, a "Missing Win Condition" message is shown. For example, Development Cost has two existing Win Conditions, hohin-winc-5 and hohin-winc-9, whereas Assurance and Usability do not have any.

For existing affected Win Conditions, the stakeholder can select them with the mouse and then click on the *Create Issue* button to have QARCC draft an Issue artifact (Screen 3). If no affected Win Conditions exist, the stakeholder can click on the *Create WinC* button to have QARCC draft a Win Condition artifact (Screen 4).

An example of the draft material produced by QARCC-1 is shown in the *Other's Comments* field of Screen 3, which cautions the stakeholders that Affordability strategies such as reuse will conflict with the Portability Win Condition if the reused software is not portable.

3. QARCC Overview

The QARCC concept of operation is described in Section 3.1, and the experimental results are summarized in Section 3.2.

3.1 QARCC Concept of Operation

Figure 8 shows the QARCC concept of operation for identifying potential quality attribute conflicts, flagging them for affected stakeholders, and suggesting options to resolve the conflicts.

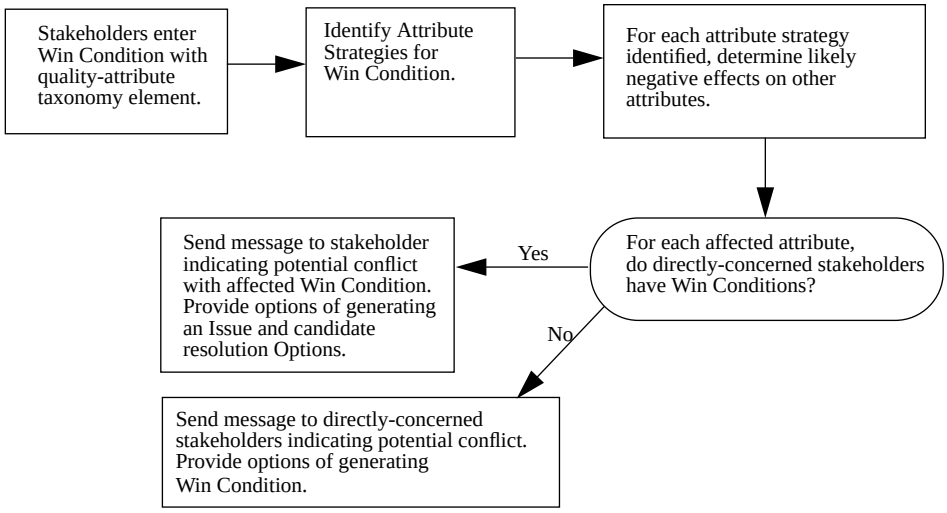


Figure 8. QARCC Concept of Operation

QARCC is triggered by a stakeholder entering a new Win Condition with a quality-attribute taxonomy element (e.g., Portability in Screen 1 of Figure 9). For this desired attribute, QARCC first considers its product and process strategies as given in Table 1 (e.g., Layering to achieve Portability). It then examines these strategies to search for potential conflicts with other attributes.

QARCC determines these potential conflicts from the portion of its knowledge base summarized in Table 2. For example, Layering produces likely conflicts with Cost/Schedule and Performance. These are shown in the Potential Conflict List in Screen 2 of

The following are the steps to build the knowledge base for the Strategy-Attribute Relations and Quality Attribute Strategies:

Step 1. Identify primitive Quality Attribute Strategies. Table 1 summarizes the current working set of strategies.

Step 2. For each identified strategy, analyze the *effect*s on each of the other primary quality attributes as simply + or -. For any pair of strategies with the same pattern of +'s and -'s, combine them if they are sufficiently synonymous.

Step 3. Define the *preconditions* and *postconditions* involved in applying the Quality Attribute Strategies. These sharpen the strategy definitions, help validate the + and - assignments, and help identify more complex interactions.

Step 4. Elaborate the more complex Strategy-Attribute Relations (not just positive/ negative). As indicated in Figure 7, the Monitoring and Control strategy improves Assurance at the cost of Performance in the near term, but also collects performance data supporting long-term Performance improvement via tuning.

Step 5. Formulate *options* to resolve the identified conflicts among quality attributes. Performance tuning is one such example.

Step 6. Update strategies based on experience.

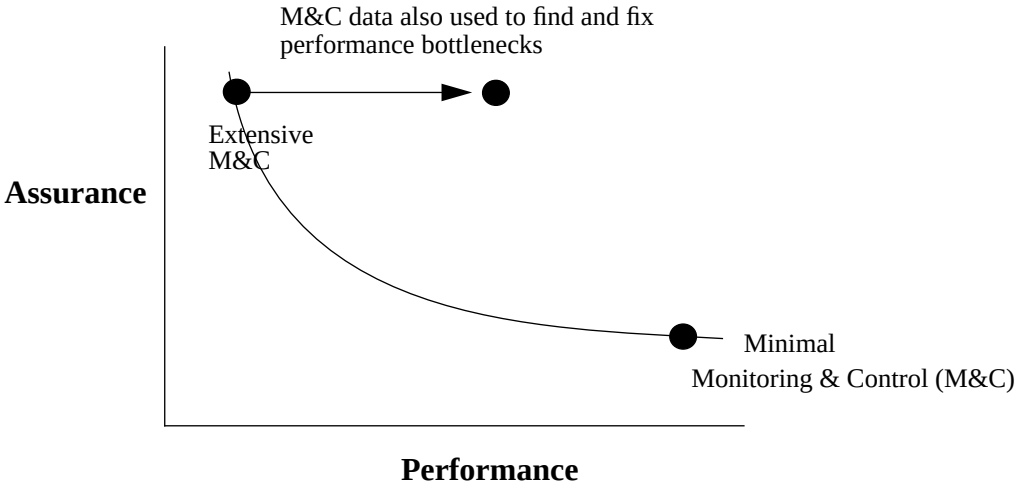
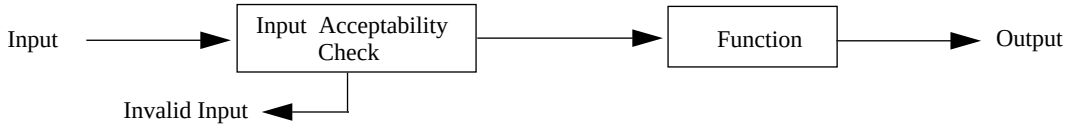


Figure 7. Performance/Assurance trades via Monitoring & Control strategy

Example 1. Input Acceptability Check:



Definition: An architectural composition that precedes a function by an acceptability check of its inputs

Preconditions:

Candidate inputs, acceptability criteria

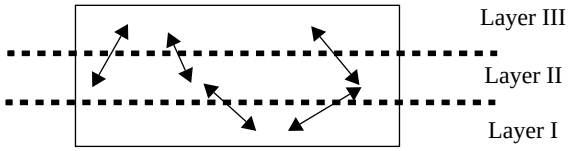
Postconditions:

If valid, pass input into the function, otherwise, indicate “Invalid Input” and exit.

Effects on quality attributes:

- Assurance: (+, unacceptable inputs filtered out)
- Performance (-, input-check requires resources)
- Cost, Schedule: (-, more to specify, develop, verify)
- Evolvability: (-, more to modify)

Example 2. Layering



Definition: A hierarchical architectural composition in which each layer can communicate only with the adjacent upwards or downwards layer

Preconditions:

Interface and protocol between a layer and an adjacent layer, request to pass data and/or control from layer to layer

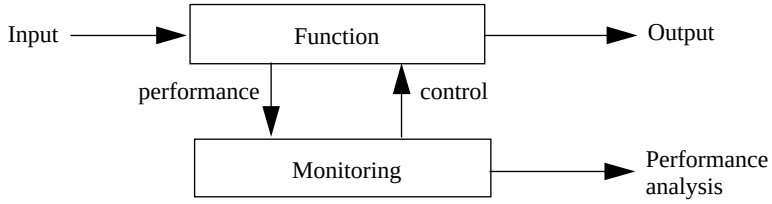
Postconditions:

Data and/or control passed from layer to layer, or notification of interface/protocol violation

Effects on quality attributes:

- Evolvability, Interoperability, Portability, Reusability: (+, hide sources of variation inside interface layers)
- Performance (-, need more interfaces, and data and/or control transfers, via protocol)
- Cost, Schedule: (-, more to specify, develop, verify)

Example 3. Monitoring & Control



Definition: An architectural composition that monitors the performance of function, controls the configuration or environment to stabilize the function (e.g., to avoid buffer overflows), and/or reports the result for subsequent performance analysis.

Preconditions:

Monitoring instrumentation, control limits and algorithms.

Postconditions:

If the function is stable, checks the performance and reports it, otherwise stabilizes the function by controlling the configuration or environment. May also report predicted future undesirable states.

Effects on quality attributes:

- Assurance: (+, avoids undesirable states)
- Performance: (-, needs additional processing in short term; +, improves performance in long term via system tuning
- Timeliness, affordability: (-, more to specify, develop, verify)

Figure 6. Three Examples of Primitive of Attribute Architecture Strategies

2.2 Elaboration of Attribute Architecture Strategies.

We have been developing a formal structure for the quality attribute architecture strategies summarized in Table 2. It is currently composed of a *definition* for each elementary strategy, *preconditions* to check whether the environments or situations are eligible for applying the strategies or not, *postconditions* to describe the results or actions after applying the strategies, *effects* on quality attributes with rationale, and *options* for resolving quality attribute conflicts.

Structure of elementary architecture strategies for quality attributes

<definition> ::= [<diagram>] Definition: <string>
<preconditions> ::= [Precondition: <string>]
<postconditions> ::= [Postcondition: <string>]
<effects> ::= [{ - <quality-attributes>: { '(' (+ / -) [, <rationale-string>] ')' } }]
<options> ::= [Options: <string>
[- Pros : <string>]
[- Cons : <string>]]

<quality-attributes> ::= { <quality-attribute> [, <quality-attribute>] }
<quality-attribute> ::= Assurance | Interoperability | Usability | Performance | Evolvability & Portability |
Cost & Schedule | Reusability

Three examples are shown in Figure 6. For example, the *preconditions* for the Input Acceptability Check strategy are sets of candidate inputs and acceptability criteria. The *postconditions* for the strategy are: if valid, pass the input into the function; otherwise, indicate "Invalid Input" and exit. The *effect* on Assurance is positive, while the *effects* on Performance, Cost/Schedule, and Evolvability are negative.

	Product Strategies	Process Strategies
Assurance	Accuracy optimization, Backup/recovery, Diagnostics, Error-reducing user input/output, Fault-tolerance functions, Input acceptability checking, Instrumentation, Integrity functions, Intrusion detection & handling, Layering, Modularity, Monitoring & Control, Redundancy	Failure modes & effects analysis, Fault tree analysis, Formal specification & verification, Inspections, Penetration, Regression test, Requirement/ design V & V, Stress testing, Test plans & tools
Interoperability	Distributability, Generality, Integrity functions, Interface specification, Layering, Modularity, Self-containedness	Interface change control, Interface definition tools, Interoperator involvement, Specification verification
Usability	Distributability (groupware), Error-reducing user input/output, Help/explanation, Modularity, Navigation, Parametrization, UI consistency, UI Flexibility, Undo, User-programmability, User-tailoring	Prototyping, Usage monitoring & analysis, User engineering, User interface tools, User involvement
Performance	Architecture balance, Descoping, Distributability, Domain architecture-driven, Faster HW, Instrumentation, Optimization (code/ algorithm), Parallelism, Pipelining, Platform-feature exploitation	Benchmarking, Modeling, Performance analysis, Prototyping, Simulation, Tuning, User involvement
Evolvability / Portability	Distributability, Generality, Input assertion/type checking, Layering, Modularity, Parameterization, Self-containedness, Understandability, User-programmability, User-tailorability, Verifiability	Benchmarking, Maintainers & user involvement, Portability vector specification, Prototyping, Requirement growth vector specification & verification
Cost / Schedule	Architecture Balance, Descoping, Domain architecture-driven, Modularity, Reuse	Design to cost/schedule, Early error elimination tools and techniques, Personnel/management, Process automation, Reuse-oriented processes, User & customer involvement
Reusability	Domain architecture-driven, Portability functions	Domain Architecting, Reuser involvement, Reuse vector specification & verification
All of Above	Descoping, Domain architecture-driven, Reuse (for attributes possessed by reusable assets)	Analysis, Continuous Process Improvement, Incentivization, Inspections, Personnel/management focus, Planning focus, Requirement/design V&V, Review emphases, Tool focus, Total Quality Management

Table 1. Quality Attribute Product and Process Strategies: General

Primary Attribute	Architecture Strategy	Other Attribute Reinforcement	Other Attribute Conflicts	Special Cases, Comments
Assurance	Input acceptability checking	Interoperability, Usability	Cost/schedule, Performance	
	Redundancy		Cost/schedule, Evolvability, Performance, Usability	
	Backup/recovery		Cost/schedule, Evolvability, Performance	
	Monitoring & Control		Cost/schedule, Performance	Performance reinforcement in long term via tuning
Interoperability	Input acceptability checking	Assurance, Usability	Cost/schedule, Performance	
.....	.....	.....	.....	.....
Evolvability/ Portability	Layering	Interoperability, Reusability	Cost/schedule, Performance	
	Modularity (Platform-Dependent Functions)	Reusability, Usability(displays)	Cost/schedule, Performance	

Table 2. Quality Attribute Strategies and Relations: Architecture Strategies

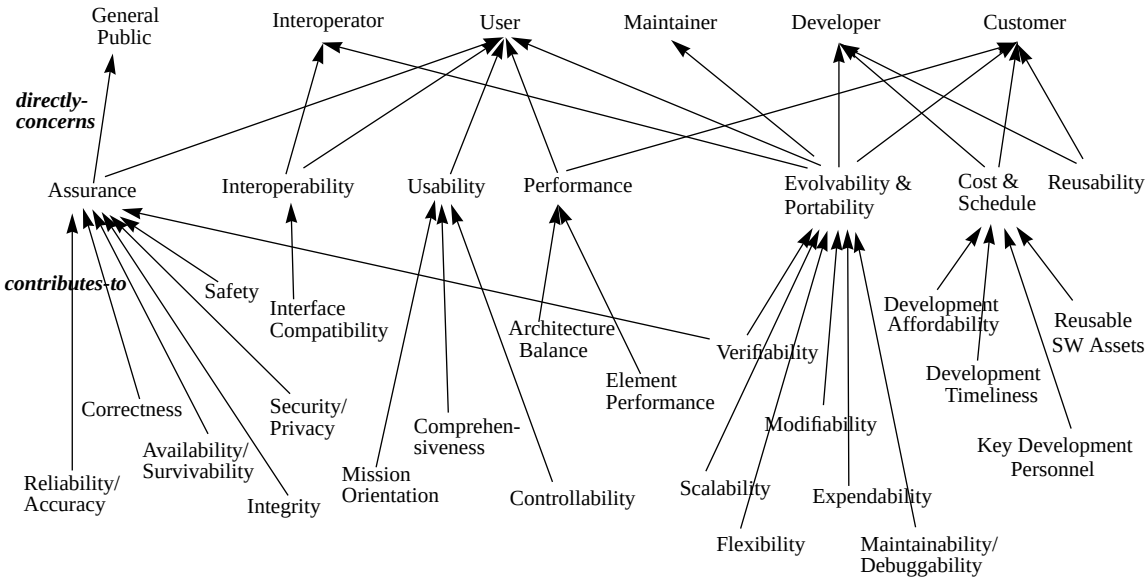


Figure 5. Mapping of Common Stakeholders' Primary Concerns onto Quality Attributes

Strategy-Attribute Relations

Table 1 shows the general set of Quality Attribute Strategies in the knowledge base, organized into product and process strategies. For example, the product strategies for improving Assurance include Input acceptability checking, Program and data redundancy, Backup and recovery, Monitoring and control, and Accuracy optimization. Process strategies for improving Assurance include Formal specification and verification, Failure modes and effects analysis, and various Test strategies.

Table 2 shows the elaboration of several example architecture-based strategies for improving quality attributes, including top-level assessments of their impact on other quality attributes. For example, the Input acceptability checking strategy applies to several Assurance sub-attributes, such as invalid data checking for Reliability and unauthorized-access checking for Security. Input acceptability checking also reinforces Interoperability via validity and access checking across system interfaces. It reinforces Usability by providing rapid feedback on invalid user inputs.

On the other hand the Input acceptability checking activities require additional code, memory, and execution cycles, and thus may conflict with the Cost/Schedule and Performance attributes.

Major components and relations of QARCC knowledge base are described in Section 2.1, and their elaboration of Attribute Architecture Strategies in Section 2.2.

2.1 Major Components and Relations

Stakeholder Roles

The most frequent roles are User, Customer, Developer, Maintainer, Interfacer, and General public. Others could be Product line managers, Testers, or Subcontractors. Complex systems may have several separate instances of roles such as Users and Customers.

First-Order Stakeholder Interests and Primary Quality Attributes

The primary attribute win conditions for each stakeholder role have been determined from experience in applying Theory W to complex, multi-stakeholder projects (e.g., Army WIS, STARS, the WinWin project itself). They are shown as the top set of arrows in Figure 5. Second-order stakeholder interests are also important (the Developer cares about Usability because the User does), but are generally addressed via negotiation of first-order stakeholder win conditions.

Attribute Elaboration and Detailed Quality Attributes

The next level of detail in the hierarchical ordering of quality attributes is shown as the lower set of arrows in Figure 5. Thus, the overall Assurance attribute, which is a primary concern of Users and the General Public, may have sub-attributes of Reliability, Accuracy, Correctness, Availability, Survivability, Integrity, Safety, Security, Privacy, and Verifiability. In many cases, it is sufficient to reason about attribute conflicts at the Primary attribute level, but at times the lower levels become important (e.g., in conflicts between fault-tolerance data distribution for Availability and restricted data access for Security).

requirements, as expressed in their Win Condition schemas. Customers may be much more concerned with Cost rather than Schedule, or vice versa. Users may be much more concerned with Performance than with Assurance, or vice versa. Overall, however, Customers' primary concerns tend to focus on such attributes as Cost and Schedule, while Users tend to be more directly concerned about such attributes as Performance and Assurance.

As indicated at the left-hand side of Figure 4, the structure of these first-order stakeholders interests forms a part of the QARCC knowledge base. It includes a quality attribute hierarchy similar to those in Basili et. al.[1], Boehm et. al.[3], and McCall et. al.[13]. Its major difference is that the highest level of the hierarchy is connected to stakeholders' primary concerns for quality attributes. As another example, Maintainers are primarily concerned with Evolvability and Portability, and only secondarily concerned with Development Affordability and Reusability, which tend to be primary concerns of Customers and Developers. This structure enables QARCC to associate quality attribute risks and conflicts with the appropriate stakeholders, in order to flag potential concerns for the stakeholders and to provide them with advice for resolving the concerns.

The other major component of the QARCC knowledge base, shown at the right-hand side of Figure 4, is a set of relationships between software architecture and process strategies and their usual effects on quality attributes. For example, a layered architecture has a positive influence on Portability contributed via a layer's ability to hide platform dependencies, whereas it has a negative influence on Performance because in-line machine dependent code may be more efficient. Thus, using a layered architecture strategy to achieve Portability will generally cause a conflict with Performance objectives.

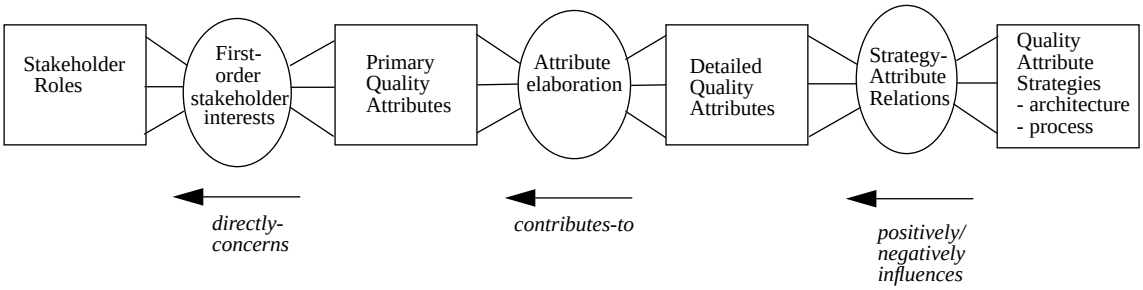


Figure 4. The QARCC Knowledge Base Structure

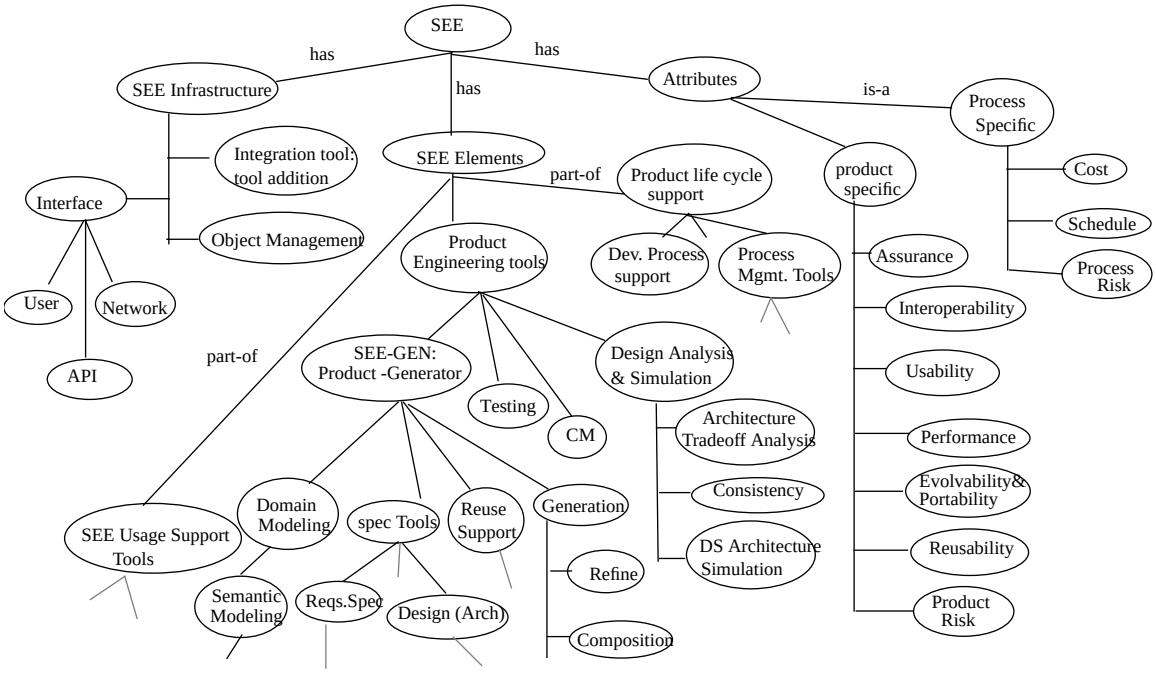


Figure 3. Example of WinWin Domain Taxonomy

For each Win Condition identifying a desired quality attribute, the QARCC tool uses a knowledge base to identify software architecture and process strategies for achieving the desired quality attribute. For each strategy, it uses another part of its knowledge base to identify potential conflicts with other quality attributes that could arise if the strategy were employed. It then provides suggestions to stakeholders on these potential quality attribute conflicts, and on options for resolving them.

The overall knowledge base structure of QARCC and its components and relations are presented in Section 2. An overview of the QARCC tool is provided in Section 3. A comparison to related work is contained in Section 4, and the conclusions in Section 5.

2. QARCC Knowledge Base Structure

The context and information available for analyzing quality attribute risks and conflicts early in the life cycle comes primarily from the system stakeholders' prioritized

mately converge on a mutually satisfactory (i.e., win-win) Option. The adoption of this Option is formally proposed and ratified by an Agreement schema, including a check to ensure that the stakeholders' iterated Win Conditions are indeed covered by the Agreement.

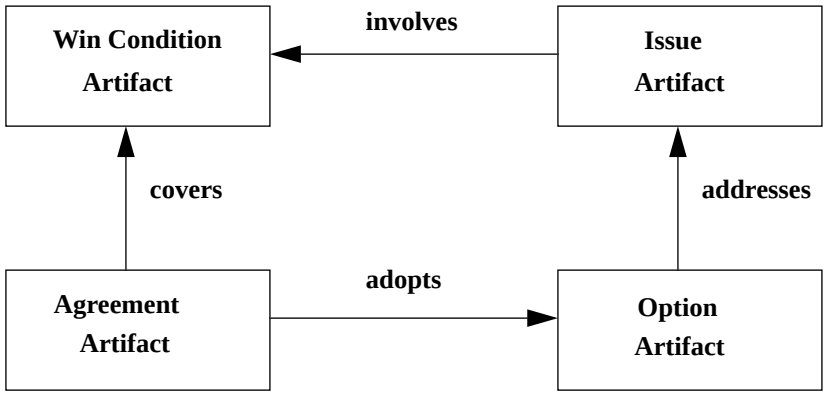


Figure 2. WinWin Negotiation Model

In large systems involving several dozen or more Win Conditions, it becomes difficult to identify conflicts among them. To aid in manual and automated conflict assessment, the Win Condition schema includes a slot for associating the Win Condition with elements of a taxonomy for the system domain being addressed.

WinWin also provides (and supports stakeholder tailoring of) domain taxonomies. An example for the Software Engineering Environment (SEE) domain is shown in Figure 3. The domain taxonomy includes two relatively domain-independent parts: the Infrastructure on the left and the Attributes on the right. It is this attribute structure that the QARCC tool uses to identify potential quality attribute conflicts among Win Conditions.

- *A capability to diagnose quality attribute conflicts based on early information*. The QARCC (Quality Attribute Risk and Conflict Consultant) tool described in this paper operates on the win conditions captured by the WinWin system to provide such a capability.

The WinWin system is based on the WinWin Spiral Model (Figure 1), which uses Theory W [2] to generate the objectives, constraints, and alternatives needed by the Spiral Model. The WinWin system assists the identified stakeholders in identifying and negotiating quality attribute conflicts, since the goal of Theory W, "Make everyone a winner," involves stakeholders identifying their quality attribute Win conditions (sector 2 in Figure 1), and reconciling conflicts among quality attribute Win conditions (sector 3).

Figure 2 shows the negotiation model used by WinWin, in terms of its primary artifacts and the relationships between them. These will be elaborated later in the paper and illustrated in Figure 9. Stakeholders begin by entering their Win Conditions, using a schema provided by the WinWin system. If a conflict among stakeholders' Win Conditions is determined, an Issue schema is composed, summarizing the conflict and the Win Conditions it involves.

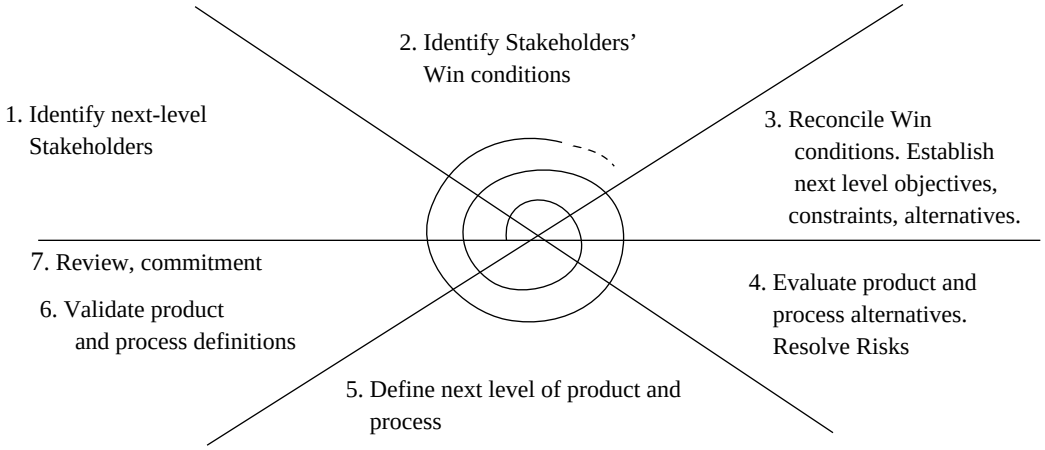


Figure 1. The WinWin Spiral Model

For each Issue, stakeholders prepare candidate Option schemas addressing the Issue. Stakeholders then evaluate the Options, iterate some, agree to reject others, and ulti-

1. Introduction

1.1 Motivation

Quite a few software projects have failed because they had a poor set of quality attribute requirements, even though they may have had well-specified functional and interface requirements. An important step in achieving successful software requirements and products is to achieve the right balance of quality attribute requirements. This involves identifying the conflicts among several desired quality attributes, and working out a satisfactory balance of attribute satisfaction. Some counterexamples illustrating the importance of achieving this balance are:

- The New Jersey Department of Motor Vehicles' licensing system, which chose a fourth generation language to satisfy software affordability and timeliness objectives, but which failed because of performance scalability problems.
- The initial design of the ARPANet Interface Message Processor software (fortunately revised), which focused on performance at the expense of evolvability through the design of an extremely tight inner loop.
- The National Library of Medicine MEDLARS II system, initially developed with a plethora of layers and recursions for portability and evolvability, but eventually scrapped due to performance problems.

The ARPANet problem was avoided by having an expert review team, which the other projects did not have. Given the overall scarcity of software expertise, it is worth trying to capture it and make it more broadly available via automated aids for software quality attribute conflict analysis.

1.2 Resolving conflicts among quality attribute requirements

At least two major capabilities are necessary to resolve conflicts among quality attribute requirements:

- *A capability to surface and negotiate conflicts and tradeoffs among quality attribute requirements*. The USC-CSE WinWin system[4,5] provides an example of such a capability.

Aids for Identifying Conflicts Among Quality Requirements

Barry Boehm and Hoh In
(boehm, hohin)@sunset.usc.edu

Computer Science Department and
Center for Software Engineering
University of Southern California
Los Angeles, CA 90089-0781

To appear in IEEE Software, March 1996 and Proceedings, ICRE 96, April 1996

Abstract

One of the biggest risks in software requirements engineering is the risk of overemphasizing one quality attribute requirement (e.g., performance) at the expense of others at least as important (e.g., evolvability and portability). This paper describes an exploratory knowledge-based tool for identifying potential conflicts among quality attributes early in the software/system life cycle.

The Quality Attribute Risk and Conflict Consultant (QARCC) examines the quality attribute tradeoffs involved in software architecture and process strategies (e.g., one can improve portability via a layered architecture, but usually at some cost in performance). It operates in the context of the USC-CSE WinWin system, a groupware support system for determining software and system requirements as negotiated win conditions.

Keywords: Software requirements engineering, Software quality, WinWin, Spiral Model, Software Architectures, Concurrent engineering, Software risk analysis, Knowledge based software engineering