

Making Practical Use of Quality Attribute Information

Ipek Ozkaya, Len Bass, and Robert L. Nord, *Software Engineering Institute*

Raghvinder S. Sangwan, *Pennsylvania State University*

Gathering and applying quality attribute information can be difficult, particularly communicating with stakeholders about quality attribute requirements and incorporating those requirements into existing analysis and design methods.

Quality attribute requirements are important both for customer and end-user satisfaction and for driving software system design. Yet asserting their importance raises many other questions. In particular, using quality attribute information in practice isn't obvious. Here, we consider two aspects of using such information: communicating with stakeholders about quality attributes and incorporating quality attribute requirements into existing analysis and design methods.

Many taxonomies and definitions of quality attributes exist. Perhaps the best known is ISO 9126, which defines 22 quality attributes and sub-attributes (which we call *quality attribute concerns*).¹ The practical questions regarding ISO 9126 are to what extent practitioners use its terminology and to what extent its quality attributes cover the qualities that concern practitioners. Data from several architectural evaluations conducted using the Software Engineering Institute's (SEI) Architecture Trade-off Analysis Method (ATAM) shows that practitioners don't use consistent terminology and that relevant taxonomies don't cover all of their concerns. To resolve terminological ambiguities, we use quality attribute scenarios to capture the stakeholders' precise concerns. This lets us supplement the terms various stakeholders use with a specification that's independent of quality attribute definitions and taxonomies.

Practitioners are insatiable. Tell them to use quality attribute scenarios to specify quality attribute requirements, and they want sufficient guidance on how to elicit these requirements that com-

plies with their development methodology. Give them that guidance, and they want to integrate the output—important quality attribute requirements—with their design method. So, we describe how practitioners can integrate quality attribute scenarios and SEI's Attribute Driven Design (ADD)² with Object-Oriented Analysis and Design (OOAD)³ to produce a design that supports both functional and quality attribute concerns.

The problem with existing approaches

Researchers and practitioners recognize the importance of eliciting quality requirements from different stakeholders' perspectives, including the system, organization, and the end-user stakeholders.⁴ However, most mainstream software design methodologies⁵ don't clearly articulate how to represent, elicit, and use quality requirements to create a software architecture. OOAD has taken center stage since the early 1980s, and almost all programming languages developed since the 1990s have object-oriented features.⁵ OOAD begins with

The Global Studio Project

This multiyear project sponsored by Siemens Corporate Research involved seven teams of 30 developers in five countries, across four continents and 11 time zones. In its first year, the project used Object-Oriented Analysis and Design; most use cases were enumerated and packaged into functionally cohesive groups. The packages were then distributed among the teams for further design and development. Because the use cases weren't fully elaborated, the teams didn't fully understand the dependencies among the various packages. As the design emerged and teams began to discover dependencies, ad hoc communication ensued between teams whose work depended on each other. New dependencies arose such that one team required a partial definition of another team's object model, sometimes at a later date. Teams didn't adequately investigate task dependencies that arose from systemic properties such as memory footprint and performance budgets, leading to duplicated work effort, conflicting solutions from other teams, and integration problems requiring rework. Coordinating and controlling work products across dependent teams became challenging, and certain teams faced design and development tasks that weren't commensurate with their skill set.

The project was undertaken from scratch in its second year using architecture-centric methods. It involved a more upfront effort focused primarily on systemic quality attributes improving the general understanding of dependencies across components that went beyond functional dependencies. As illustrated in figure 3 in the main article, the system's structure was primarily driven by quality attributes. The top-level system was initially assigned responsibilities embodied in the use cases. As the top-level system was decomposed, its responsibilities were distributed across newly created components. This gave a better understanding of dependencies (both functional and systemic) across components, which were then considered when distributing the work across teams, resulting in significantly improved coordination and control of work products across dependent teams.

use cases and primarily uses functional decomposition to drive the system's architecture. (The running example we used in this article to demonstrate the use of OOAD and architecture-centric methods is based on the Global Studio Project [see the sidebar].⁶ You can find a survey of other design methods elsewhere.⁷)

Consider a company that builds hardware-based field systems for controlling a building's internal functions, such as heating, ventilation, air conditioning, access, and safety. Hardware's commoditization has led to shrinking profit margins. To improve the margins, the company wants to develop a software system to automatically monitor and control the building's internal functions. The system users would be facilities managers, and the system would broadly perform the following functions:

- Manage a network of hardware-based field systems used for controlling building functions.
- Issue commands to configure the field systems.
- Define rules on the basis of property values of

field systems that trigger reactions and issue commands to reset these property values.

- Trigger alarms notifying appropriate users of life-critical situations.

The company wants to offer this product in new and emerging geographic markets and expand its sales channels by letting value-added resellers sell the software system under their own brands. The resellers would support field systems from manufacturers they choose.

Figure 1 illustrates the various artifacts an OOAD application would produce if it were used to develop this product.⁸ The use cases broadly explain the functionality the product must support (figure 1a). Figure 1b shows the building automation problem's domain model, illustrating the important business entities that the use cases manipulate while they execute. Because the use cases capture the interaction of external entities at a system's boundary, they help define the system interfaces (figure 1c). These business entities define business interfaces (figure 1d) that they must provide to the use cases. Typically, the architect creates a system interface for every use case and a business interface for every core business entity, shown with a <<core>> stereotype in the UML model. (A core business entity is one with no mandatory association.⁸) The architect groups semantically coherent system interfaces, which become the responsibility of a single system component (figure 1e), whereas a business component is created for each core business entity (figure 1f). Together, figures 1e and 1f define the system's architecture.

This approach is use-case driven with little focus on the quality attributes. For instance, this architecture doesn't explicitly accommodate the quality attribute requirements associated with supporting new and emerging geographic markets and different value-added resellers. These requirements call for modifiability concerns such as adding a new field system or supporting a new language. However, without further refining the architecture (for example, introducing adaptors and factories), making such changes later in the development life cycle would be costly.

To some extent, the Rational Unified Process overcomes OOAD's lack of focus on quality attributes. RUP, a software development process widely used with OOAD,⁹ divides the software development life cycle into four phases—inception, elaboration, construction, and transition. During the elaboration phase, the architect creates a baseline executable architecture by focusing on architecturally significant use cases chosen on the basis of risk,

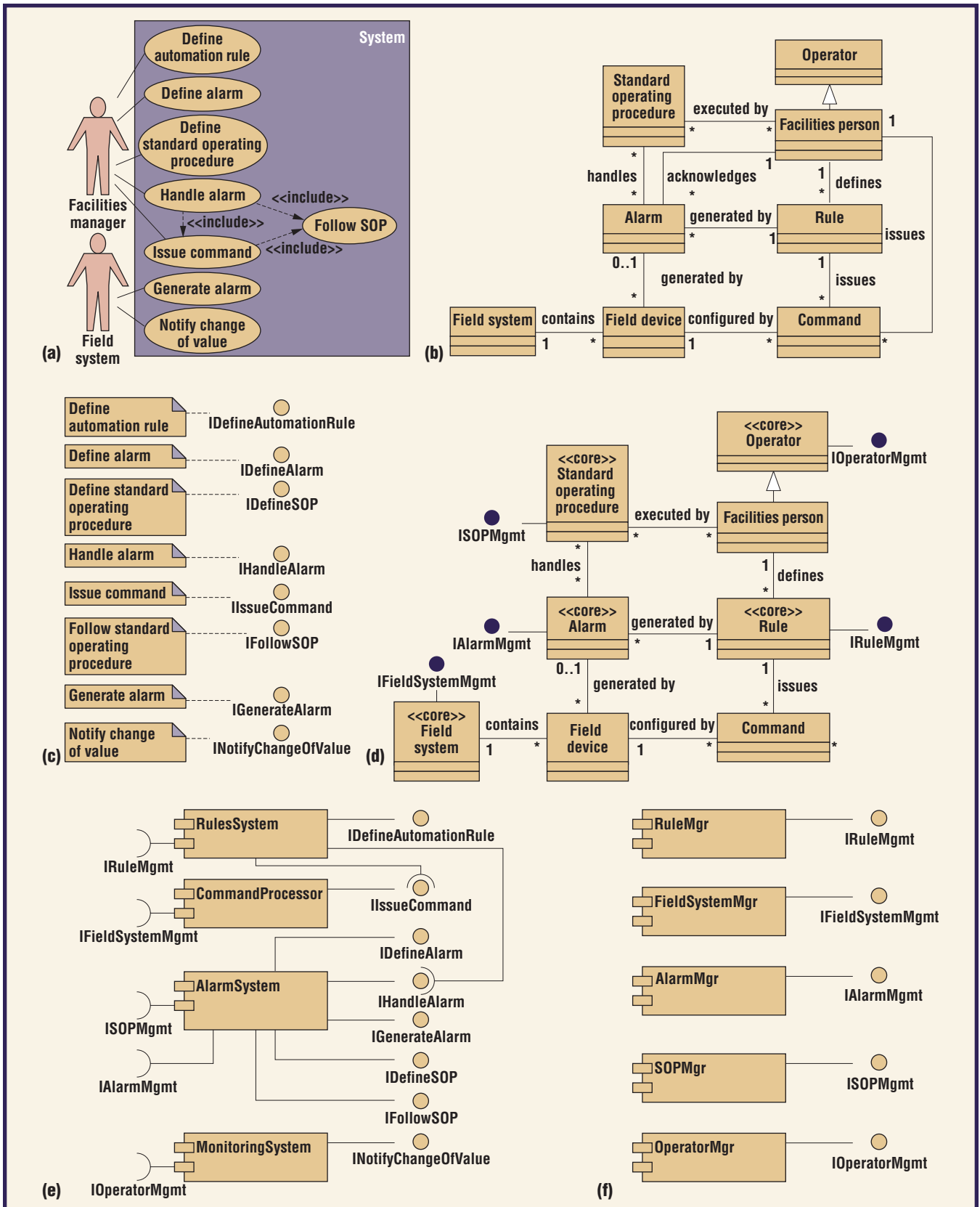


Figure 1. Object-oriented analysis and design artifacts showing (a) use cases, (b) the business domain model, (c) system interfaces, (d) business interfaces, (e) system components, and (f) business components. (The figure uses UML notation.)

Table 1**The most common quality attributes according to the SEI-led ATAM evaluations.**

Quality attribute	Distribution (%)
Modifiability	14.1
Performance	13.6
Usability	11.4
Maintainability	8.5
Interoperability	7.8
Security	7.3
Configurability	6.9
Availability	6.8
Reliability	5.7
Scalability	3.2
Testability	2.6
Affordability	2.0
Reusability	1.9
Integrability	1.9
Safety	1.1
User data management	1.0
Portability	0.8
Assurance	0.8
Product line	0.8
Net-centric operation	0.5

criticality, and coverage.^{3,9} A use case is architecturally significant if it

- poses technical risks such as resource contention or performance;
- is business critical, capturing the system's essence; or
- exercises most of the system.

Although the business-critical use cases might be clear at the beginning of elaboration, identifying those that pose a technical risk or that exercise most of the system can be difficult. Not all use cases are fully understood until the end of elaboration, and the system exists only as an evolutionary prototype.

Because it might be hard to express certain quality attribute requirements as use cases, RUP also maintains a supplementary specification document for such requirements. In addition to the architecturally significant use cases, this document's quality attribute requirements also guide the architectural efforts. In practice, RUP offers no guidance

on representing quality attribute requirements. The supplementary specification document can become unwieldy, acting as a sink for every conceivable quality attribute requirement that might apply to the system under consideration.

Discussing quality concerns with stakeholders

The SEI has been using quality attribute scenarios in their architecture-centric methods to better elicit and represent quality attribute requirements observed in practice. A *quality attribute scenario* is a quality attribute specific requirement with six parts: the stimulus, its source, the environment, the artifact, the response, and the response measure.² It provides an operational definition for a quality attribute. For example, it's meaningless to say that a system is modifiable. Every system is modifiable with respect to one set of changes and not modifiable with respect to another. It's more meaningful to cast the requirement as a scenario, such as

A developer wishes to add an input field to the UI code at design time; modification is made with no side effect in three hours.

These scenarios can help structure the supplemental requirements of a system under development using OOAD.

During one step of the ATAM,² the evaluators ask the system's stakeholders to construct a utility tree. The utility tree has as its leaves a collection of quality attribute scenarios. However, what's more relevant for this discussion is that its second and third levels are, respectively, a collection of quality attributes and their concerns (subattributes in the usage of ISO 9126). During the ATAM evaluation, the stakeholders are encouraged to use their own language for quality attributes. Thus, the utility trees that SEI collected during ATAM evaluations became a good resource for studying the language stakeholders used to describe quality attributes.

We examined the utility tree data from 24 different SEI-led evaluations from 1999 to 2007. These evaluations, conducted for commercial and government organizations, range from avionics to transportation to combat systems. The data has 1,072 scenarios in total. We looked at trends in the quality attributes and quality attribute concerns, looking for matches and mismatches between stakeholders' language, the quality attributes that concern them, and the existing guidance for collecting and specifying quality attributes.

Table 1 shows the 20 most common quality attributes (out of 49 attributes) in our data. The

distribution percentage represents the number of times the concept is expressed as a top-level quality attribute, divided by the total number of scenarios. To understand the true nature of stakeholder concerns, we also aggregated the quality attribute scenarios by their specified quality attribute concern level as opposed to the quality attributes reported in table 1. Each quality attribute in our data has between two and about a dozen concerns. Table 2 lists the top 20 out of 140 concerns.

Many of the top 20 quality attributes or concerns don't appear in the same fashion in common taxonomies. For example, ISO 9126 defines security as a subattribute of functionality: "The capability of the software product to protect information and data so that unauthorized persons or systems cannot read or modify them and authorized persons or systems are not denied access to them." This is clearly a functional concern. However, other security concerns must also be captured for requirements to provide clear guidance for design. In our data, security is a top-level quality attribute with 16 concerns or subattributes (see table 3). Although authentication and access control (as described in ISO 9126) made up 27 percent of the quality attribute concerns in our data, there were several other concerns such as data integrity and malicious code. We observe similar differences with the FURPS+ scheme as well.¹⁰ (FURPS+ stands for functionality, usability, reliability, performance, and supportability, and the + indicates other possible attributes.)

Without the corresponding quality attribute scenarios, ambiguities are inevitable, and these categories can easily become misleading, especially when communicating with stakeholders. As the "Dialogue with a Stakeholder" sidebar demonstrates, stakeholders have their own meaning for quality attribute concerns such as flexibility. Although the sidebar is anecdotal, it is representative of our data. In our data, flexibility is a concern of several quality attributes—namely interoperability, configurability, and extensibility. To infer the quality concern under such circumstances, you must examine the scenario. About half of these flexibility scenarios refer to changing the software, and the rest refer to the software's ability to cover several usage conditions. However, we collected these ambiguous classifications from stakeholders during ATAM evaluations. Our data doesn't distinguish between stakeholders who have investigated various standardized taxonomies and those who haven't.

We observed that stakeholders' descriptions of quality attributes didn't follow any pattern relative to various standard descriptions of quality attributes. When specifying quality attributes, it's bet-

Table 2

The most common quality attribute concerns according to the SEI-led ATAM evaluations

Quality attribute	Quality attribute concern	Distribution (%)
Modifiability	New/revised functionality/components	6.4
Usability	Operability	4.1
Modifiability	Upgrade/add hardware components	3.9
Performance	Performance response time/deadline	3.6
Performance	Performance latency	3.2
Modifiability	Portable to other platforms	3.1
Interoperability	Operate intraservice (for example, ship-to-ship)	2.8
Usability	Usability ease of operation: can do within a time limit	2.7
Performance	Throughput	2.1
Performance	Resource utilization	1.9
Availability	Failure recovery/containment	1.9
Configurability	Flexibility (range of operation scenarios)	1.7
Availability	Graceful degradation	1.6
Interoperability	Compliance to standards/protocols	1.5
Affordability	Affordability of various decisions	1.5
Modifiability	Replace COTS	1.4
Performance	Real time	1.4
Availability	Fault tolerance	1.3
Configurability	Discovery (new configuration)	1.3
Security	Authentication	1.3

ter to elicit the concern supported by the quality attribute scenario description than to go down a list of classifications, which might not cover all quality issues.

Building quality into software projects

Returning to the example of the company that wants to build a software system to monitor a building's internal functions, you might note that the forces that most significantly impact its architecture are its business goals. The company also would have to consider different geographic markets' languages, cultures, and regulations, and they would have to integrate field systems from different manufacturers into the product to meet the value-added resellers' needs. The company would have to make trade-offs and perhaps scale back if they can't afford the business goals' inherent risks. (We use the term *business goals* differently from its use in the business modeling context.) You can use busi-

Table 3**The distribution of security-related quality attribute concerns according to the SEI-led ATAM evaluations**

Quality attribute concern	Distribution (%)
Authentication	16.9
Multilevel security	13.0
Access control	10.4
Ability to change security policy	9.1
Data integrity	6.5
Intrusion detection	6.5
Confidentiality	5.2
Data identification	5.2
Data protection	5.2
Blocking	5.2
Sanitization	5.2
Accountability	3.9
Service disruption	2.6
Malicious code	2.6
Denial of service	1.3
Migration of security in later release	1.3

Table 4**Business goals used for eliciting quality attribute scenarios**

Business goal	Goal refinement	Quality attribute
Expand sales channels through value-added resellers	Support field systems from different manufacturers	Modifiability
	Support conversions of nonstandard units used by the different field systems	Modifiability
Enter new and emerging geographic markets	Support several international languages	Modifiability
	Support regulations that require life-critical systems, such as fire alarms, to operate within specific latency constraints	Performance

ness modeling to engineer new business processes or reengineer existing ones. Recent extensions to the RUP provide support for business modeling.¹¹

SEI's architecture-centric methods use business goals to elicit their associated quality attribute scenarios, which ADD then uses to create a software

architecture. Stakeholders provide key input when eliciting quality attribute scenarios, but not all stakeholders participate in architecting the system. Table 4 shows the business goals and subgoals and their associated quality attributes, which the stakeholders use to elicit quality attribute requirements for the building automation system.

Often, fewer than a dozen architecturally significant requirements drive or shape the architecture² as opposed to the hundreds of functional requirements. So, when it comes to understanding quality attributes, coverage is most critical. Organizing quality attribute scenarios into a utility tree provides significant coverage of the quality attribute requirements in a short amount of time. Figure 2 shows a portion of the utility tree for the building-automation system that systematically fleshes out the modifiability and performance quality attributes into concrete quality attribute scenarios. The quality attribute concerns trace back to table 4's business subgoals.

The ADD approach to defining software architecture bases the design process on the system's quality attribute requirements. ADD helps designers understand quality attribute trade-offs early in the design process. The method takes the scenarios (such as those in figure 2) and prioritizes them before the design process can begin. In the prioritization tuple next to the quality attribute scenario, the letters represent priorities that are high, medium, or low. The tuple's first letter is the priority that the business stakeholders generated to represent the quality attribute's business value. The second letter is the priority that the technical stakeholders generated to represent the effort required to achieve the quality attribute. For instance, (H, H) means that the quality attribute has high business value and would require considerable effort to achieve. ADD considers the quality attributes in the following order: (H, H), (H, M), (H, L), (M, H), (M, M), (M, L), (L, H), (L, M), and (L, L). So, the final architecture created through such a process also reflects this priority order. Notice the distinction between the architecture in figure 3 and the one derived from an OOAD application in figures 1e and 1f.

ADD starts with the top-level system (figure 3a) and decomposes it into components (figure 3b–3d), applying architectural tactics² corresponding to the quality attributes. Relevant tactics are applied based on the priorities shown in the utility tree in figure 2.

Figure 3b shows the application of the modifiability tactics to limit the impact of change and minimize the number of dependencies on the system responsible for integrating new field systems.

Dialogue with a Stakeholder

Evaluator: What are some of your main concerns for the system being evaluated?

Stakeholder: We have to provide flexibility.

Evaluator: What do you mean by "flexibility"?

Stakeholder: Well, flexibility has two thrust areas for our product, one from the user perspective and one from the system perspective.

Evaluator: Are they related?

Stakeholder: Yes and no.

Evaluator: Please give an example from both the user perspective and the system perspective where the system should address flexibility concerns.

Stakeholder: When we talk about flexibility from the user's viewpoint, the users should be able to do everything that they used to do before this system was deployed, either using the new screen layout or switching to the old screen layout.

Evaluator: Is it reasonable to say that, from the end users' perspective, you would like the user interfaces to be customizable, allowing switching to earlier versions?

Stakeholder: Absolutely.

Evaluator: How about system flexibility?

Stakeholder: The system shouldn't constrain the data flow. If a particular flow of information doesn't work well, we should be able to create a new process.

Evaluator: So, the system shouldn't provide one predetermined process but should let you create new processes or modify existing processes.


Stakeholder: Exactly. We would like the system to be flexible rather than constraining.

We introduce an adapter for each field system (*anticipation of changes* tactic) with each adapter exposing a standard interface (*maintain existing interface* tactic). We also introduce a virtual field system to further limit the ripple effect when removing or adding field systems (*introducing an intermediary* tactic).

Figure 3c illustrates the application of the performance tactic (*concurrency*) to support critical systems so they operate within specific latency constraints. We separate the parts responsible for evaluating rules and generating alarms for life-threatening situations into logic-and-reaction and alarm modules. We can then move these modules to a dedicated execution node, which reduces latency, and can further enhance its performance by introducing multithreading in these modules.

Figure 3d shows the application of the modifiability tactic (*anticipation of changes*). We create a separate presentation module to support several international languages.

The only scenario from figure 2 that we don't appear to address is the one dealing with converting nonstandard units that various devices use. We use the adapters in figure 3b to do the conversions into standard units (introducing an *intermediary* tactic).

Business goals and their associated quality attribute requirements strongly influence a system's architecture. However, eliciting and representing the quality attribute information hasn't yet become a routine part of practice. Although taxonomies are helpful, without quality attribute scenarios it's often impossible to infer a quality attribute's essence, especially where a quality attribute scenario covers multiple quality attributes. Integrating architecture-centric methods, such as ADD, into mainstream software development methodologies produces significant benefits.^{12,13} Although it's not yet routine, some organizations are taking steps forward. Organizations such as Pitney Bowes have defined company-specific usage of quality attributes and their concerns, resulting in less confusion about the meaning of quality attribute descriptions among the stakeholders.¹⁴ Furthermore, Ericsson, Sandia National Laboratories, and Samsung SDS have effectively incorporated quality attribute information into existing methods.¹⁵ 

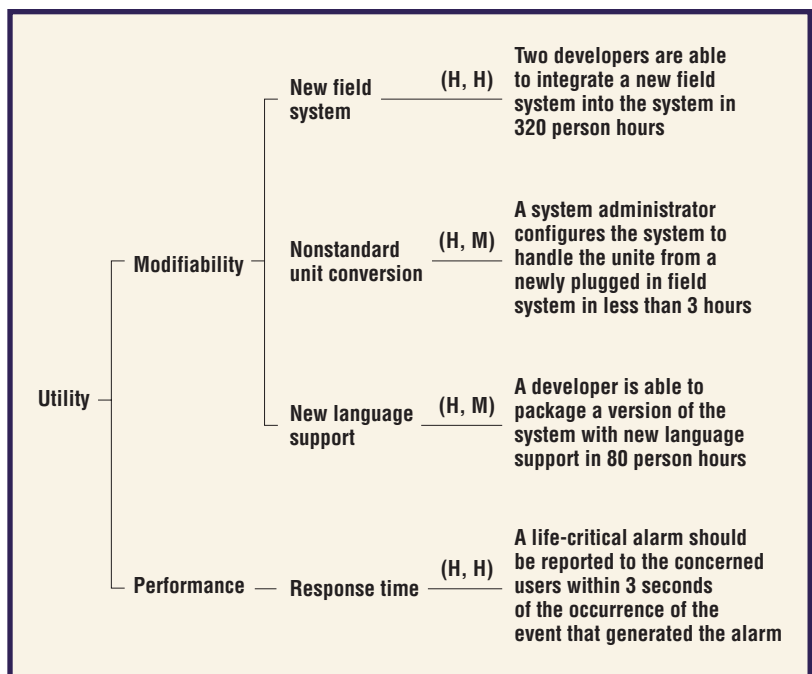
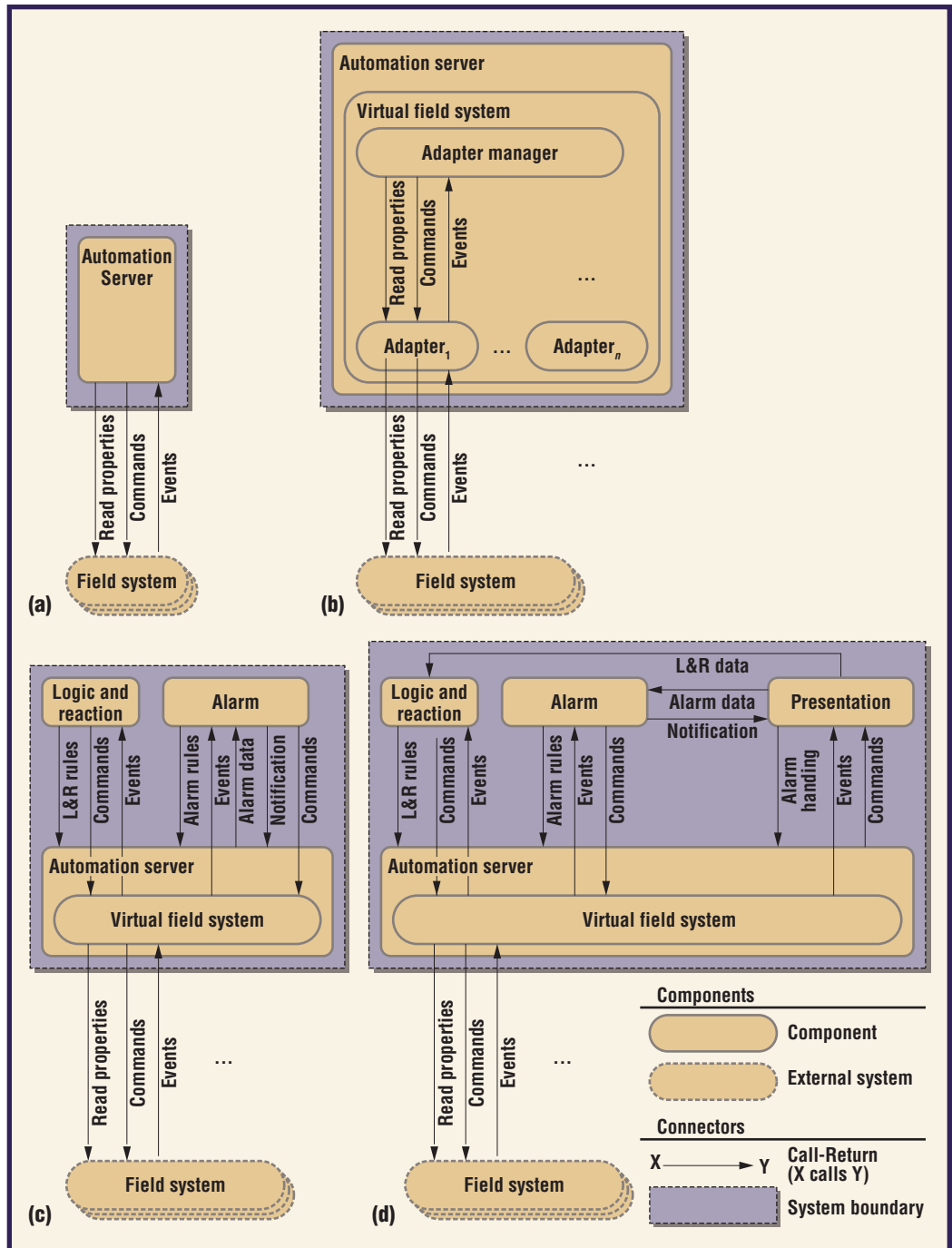


Figure 2. Utility tree for the building automation system. The letters represent high, medium, or low priorities for the business and technical stakeholders, respectively.

References

1. ISO 9126-1 Information Technology—Software Product Evaluation—Quality Characteristics and

Figure 3. Attribute-driven design showing (a) the top-level system as a starting point and its decomposition through applying tactics to support (b) adding new field systems, (c) latency constraints, and (d) internationalization.



Guidelines for Their Use, Int'l Organization for Standardization, 2001.

- L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 2nd ed., Addison-Wesley, 2003.
- C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd ed., Prentice Hall, 2004.
- K. Wiegers, *Software Requirements*, 2nd ed., Microsoft Press, 2003.
- D. Budgen, *Software Design*, Addison-Wesley, 2003.
- N. Mullick et al., "Siemens Global Studio Project: Experiences Adopting an Integrated GSD Infrastructure," *Proc. IEEE Int'l Conf. Global Software Eng. (ICGSE 06)*, IEEE Press, 2006, pp. 203–212.
- C. Hofmeister et al., "A General Model of Software Architecture Design Derived from Five Industrial Approaches," *J. Systems and Software*, vol. 80, no. 1, 2007, pp. 106–126.
- J. Cheesman and J. Daniels, *UML Components: A Simple Process for Specifying Component-Based Software*, Addison-Wesley, 2001.
- P. Kroll and P. Kruchten, *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP*, Addison-Wesley, 2003.
- P. Eeles, "Capturing Architectural Requirements," *IBM DeveloperWorks*, 15 Nov. 2005, www-128.ibm.com/developerworks/rational/library/4706.html.
- H. Eriksson and M. Penker, *Business Modeling with*

PURPOSE: The IEEE Computer Society is the world's largest association of computing professionals and is the leading provider of technical information in the field. Visit our Web site at www.computer.org.

EXECUTIVE COMMITTEE

President: Rangachar Kasturi*
President-Elect: Susan K. (Kathy) Land; * **Past President:** Michael R. Williams; * **VP, Electronic Products & Services:** George V. Cybenko (1ST VP); * **Secretary:** Michel Israel (2ND VP); * **VP, Chapters Activities:** Antonio Doria; † **VP, Educational Activities:** Stephen B. Seidman; † **VP, Publications:** Sorel Reisman; † **VP, Standards Activities:** John W. Walz; † **VP, Technical & Conference Activities:** Joseph R. Bumblis; † **Treasurer:** Donald F. Shafer; * **2008–2009 IEEE Division V Director:** Deborah M. Cooper; † **2007–2008 IEEE Division VIII Director:** Thomas W. Williams; † **2008 IEEE Division VIII Director-Elect:** Stephen L. Diamond; † **Computer Editor in Chief:** Carl K. Chang†

* voting member of the Board of Governors

† nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 2008: Richard H. Eckhouse, James D. Isaak, James W. Moore, Gary McGraw, Robert H. Sloan, Makoto Takizawa, Stephanie M. White
Term Expiring 2009: Van L. Eden, Robert Dupuis, Frank E. Ferrante, Roger U. Fujii, Ann Q. Gates, Juan E. Gilbert, Don F. Shafer
Term Expiring 2010: André Ivanov, Phillip A. Laplante, Itaru Mimura, Jon G. Rokne, Christina M. Schober, Ann E.K. Sobel, Jeffrey M. Voas

Next Board Meeting:

16 May 2008, Las Vegas, NV, USA

EXECUTIVE STAFF

Executive Director: Angela R. Burgess; **Associate Executive Director:** Anne Marie Kelly; **Associate Publisher:** Dick Price; **Director, Administration:** Violet S. Doan; **Director, Finance & Accounting:** John Miller

COMPUTER SOCIETY OFFICES

Washington Office. 1828 L St. N.W., Suite 1202, Washington, D.C. 20036-5104

Phone: +1 202 371 0101 • Fax: +1 202 728 9614
 Email: hq.ofc@computer.org

Los Alamitos Office. 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1314

Phone: +1 714 821 8380 • Email: help@computer.org

Membership & Publication Orders:

Phone: +1 800 272 6657 • Fax: +1 714 821 4641

Email: help@computer.org

Asia/Pacific Office. Watanabe Building, 1-4-2 Minami-Aoyama, Minato-ku, Tokyo 107-0062, Japan

Phone: +81 3 3408 3118 • Fax: +81 3 3408 3553

Email: tokyo.ofc@computer.org

IEEE OFFICERS

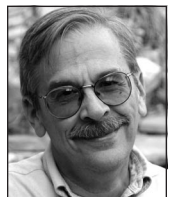
President: Lewis M. Terman; **President-Elect:** John R. Vig;
Past President: Leah H. Jamieson; **Executive Director & COO:** Jeffrey W. Raynes; **Secretary:** Barry L. Shoop;
Treasurer: David G. Green; **VP, Educational Activities:** Evangelia Micheli-Tzanakou; **VP, Publication Services & Products:** John Baillieul; **VP, Membership & Geographic Activities:** Joseph V. Lillie; **VP, Standards Association Board of Governors:** George W. Arnold;
VP, Technical Activities: J. Roberto B. deMarca; **IEEE Division V Director:** Deborah M. Cooper; **IEEE Division VIII Director:** Thomas W. Williams; **President, IEEE-USA:** Russell J. Lefevre

About the Authors



Ipek Ozkaya is a senior member of the technical staff in the Product Line Systems Program at the Software Engineering Institute at Carnegie Mellon University, where she works in the Software Architecture Technology initiative. Her research interests include developing methods for improving software architecture practices focusing on software economics and requirements management. She received her PhD in computational design from Carnegie Mellon University. Contact her at 4500 Fifth Ave., Pittsburgh, PA 15213; ozkaya@sei.cmu.edu.

Len Bass is a senior member of the technical staff in the Product Line Systems Program at the Software Engineering Institute at Carnegie Mellon University. His research interests include techniques for methodically designing software architectures, better supporting usability through software architecture, and understanding the relationship between software architecture and global software development practices. He received his PhD in computer science from Purdue University. He coauthored *Documenting Software Architecture: Views and Beyond* (Addison-Wesley, 2002) and *Software Architecture in Practice* (Addison-Wesley, 2000). Contact him at 4500 Fifth Ave., Pittsburgh, PA 15213; ljb@sei.cmu.edu.



Raghvinder S. Sangwan is an assistant professor of information science in Pennsylvania State University's Great Valley School of Graduate Professional Studies. His teaching and research involve analysis, design, and development of software systems and their architecture, as well as automatic and semiautomatic approaches to assessing their design and code quality. He received his PhD in computer and information sciences from Temple University. Contact him at 30 E. Swedesford Rd., Malvern, PA 19355; rsangwan@psu.edu.

Robert L. Nord is a senior member of the technical staff in the Product Line Systems Program at the Software Engineering Institute at Carnegie Mellon University. His work involves developing and communicating effective software architecture methods and practices. He received his PhD in computer science from Carnegie Mellon University. He coauthored *Documenting Software Architecture: Views and Beyond* (Addison-Wesley, 2002) and *Applied Software Architecture* (Addison-Wesley, 2000). He's a member of the ACM and IFIP Working Group 2.10 Software Architecture. Contact him at 4500 Fifth Avenue, Pittsburgh, PA 15213; rn@sei.cmu.edu.

- UML: *Business Patterns at Work*, OMG Press, 2000.
12. R. Kazman et al., *Integrating Software Architecture-Centric Methods into the Rational Unified Process*, tech. report CMU/SEI-2004-TR-011, SEI, Carnegie Mellon Univ., 2004.
 13. R.L. Nord and J.E. Tomayko, "Software Architecture-Centric Methods and Agile Development," *IEEE Software*, vol. 23, no. 2, 2006, pp. 47–53.
 14. R.L. Nord et al., *Proc. 1st Software Architecture Technology User Network (SATURN) Workshop*, tech. note CMU/SEI-2005-TN-037, SEI, Carnegie Mellon Univ., 2005.
 15. *Proc. 3rd SEI Software Architecture Technology User Network Workshop (SATURN 07)*, SEI, Carnegie Mellon Univ., 2007, www.sei.cmu.edu/architecture/saturn/2007/tech_program.html.