

Lessons from Developing Nonfunctional Requirements for a Software Platform

Xiping Song, Beatrice Hwong, and Johannes Ros,
Siemens Corporate Research

// Lessons learned in developing NFRs for a large, shared software platform include techniques to help manage NFRs and automate platform performance testing. //



A SOFTWARE PLATFORM is a component set that provides similar services to different applications. When the platform is shared, it enables software reuse across multiple applications.¹ Shared platforms can apply service-oriented architecture (SOA) techniques, such as Web services and e-commerce hosting, and they can be specific to a technical

domain, such as messaging middleware, or to an application domain, such as medical image processing.

Recently, we worked on a shared, service-oriented software platform that was technical-domain-specific across multiple application domains, including transportation, industrial control, telecommunications, and others. There

were about eight major stakeholders, each representing a Siemens business unit, and they had nearly every kind of nonfunctional application requirement for safety, performance, interoperability, and so on. The project employed hundreds of developers, including a dozen requirements engineers, located at multiple global sites.

Our task was to complete the nonfunctional requirements (NFR) specification. We inherited some draft NFR specifications and stakeholder inputs in the form of responses to requirements elicitation questionnaires and other documents that defined NFRs for similar, prior products. Our specific goals were to address three issues that were delaying the project:

- The draft NFR specifications required the platform to support some very large capacities. In many cases, architects didn't know whether such capacities were feasible or why they had to be so large.
- The quality assurance team faced difficulties specifying the performance test plan because they didn't necessarily know the deployment configurations for each performance requirement.
- Architects and other engineering teams had trouble interpreting the draft NFR specifications because the different targeted application domains often gave different meanings to the same term.

After working on the NFRs for several weeks, we understood the root causes of these issues:

- The platform NFRs dealt with a large number of deployment variations, which complicated the context and rendered the NFRs untestable.
- The platform NFRs dealt with a

large number of load- and resource-sharing conditions, further complicating the NFR conditions.

- Most of the platform NFRs were based on the nonfunctional behaviors of existing products. Because a platform-based product implements functionalities through a combination of product features and platform services, requirements engineers can only indirectly derive the platform service NFRs from uncertain and incomplete product information.

Our analysis is detailed elsewhere.² Here, we briefly review the process we defined to address these issues and the lessons we learned in applying it.

The Platform NFR Development Process

We defined the Platform NFR Development (PND) process to help systematically develop a high-quality NFR specification using structured NFR artifacts. PND focuses on making the NFR specification complete, consistent, verifiable, and traceable, as defined by IEEE.³

Figure 1 depicts the process. In the following descriptions, we focus only on those activities most relevant for handling platform NFRs. Although we present the process in a procedural paradigm, the actual execution of PND activities is likely to be parallel and iterative.

Define Questionnaires and Elicit Stakeholder Inputs

We obtained stakeholder input in two rounds of activity. First, the stakeholders answered questionnaires with their best knowledge, and then refined their answers in a workshop with the requirements engineers.

The first round aims to establish a basic understanding of stakeholder NFRs. Using questionnaires ensures a consistent scope for all stakeholders.

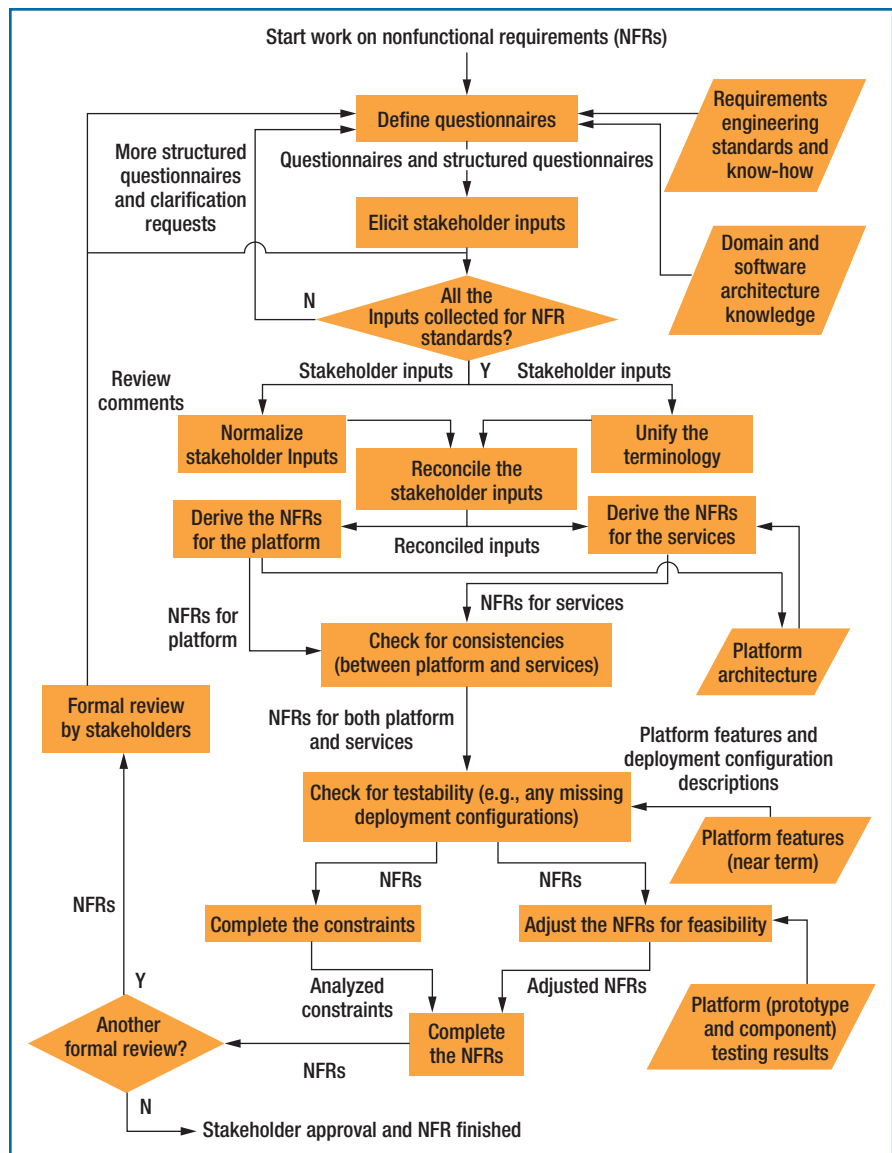


FIGURE 1. The Platform NFR Development (PND) process. PND is based on IEEE-recommended practices for complete, consistent, verifiable, and traceable specification of software requirements (IEEE Std 830-1998).

Industrial standards, such as ISO 9126 on quality attributes, provide a good basis for defining the questionnaires.

The requirements engineers use the answers to create structured tables to support more detailed NFRs in the second round. For example, to the question “What performance requirements do you have?” a stakeholder might answer, “The large deployment of our

system needs to process 1,000 requests per second.” Requirements engineers can capture specific values for “large deployment” and “requests” in a structured table. This ensures more complete NFR definitions because it constrains stakeholders with specific loads and deployment configurations and focuses on collecting quantifiable inputs.

After entering quantifiable inputs

TABLE 1

Merged questionnaire example.

Stakeholder ID	Number of servers	
	Small deployment	Large deployment
Stakeholder 1	3	4
Stakeholder 2	1	6
Average	2	5

per stakeholder into spreadsheets, we merge them into one spreadsheet that groups the answers. When stakeholders use different domain-specific terms and metrics, we perform the next two activities, “Unify terminology” and “Normalize stakeholder inputs,” on the grouped answers.

Table 1 provides a simple example showing the number of servers needed for specific deployment sizes. Our recent work provides more details about the table.⁴ Eliciting other NFRs, such as interoperability and security, might require accommodating more textual explanations than structured questionnaire responses.

Unify Terminology

Because a shared platform supports multiple application domains, different stakeholders are likely to use terms differently.⁵ For example, “alarms,” “events,” “requests,” and “value change events” can all represent data items that are similar from the platform point of view but different in various applications. To unify the terminology in this case, for example, we adopted the generic term “message” in all the contexts where the application aspects weren’t significant.

Normalize Stakeholder Inputs

It helps to make different stakeholders’ inputs to platform NFRs directly comparable in scale and operating conditions. For example, one stakeholder’s input might specify “processing 50

alarms per 10 seconds,” while another’s input specifies “processing 20 alarms/second.” Such differences are common when performance requirements are derived from different application domains or legacy products.

To make the inputs directly comparable, requirements engineers must convert them to the same scale (for example, alarms/second). Sometimes, this normalization changes a stakeholder’s original intent. For example, “Process 50 alarms per 10 seconds” as a product performance requirement expresses a more precise need than a processing rate specified in alarms per second. Normalizing it to “Process 5 alarms/second” makes it a more demanding NFR request.

Reconcile Stakeholder Inputs

To address a group of similar, normalized stakeholder inputs, requirements engineers must define one or more NFRs. To do so, they identify a range of variations for similar requirements. If the range is wide, they might add some constraints to ensure that the NFRs are feasible to implement.

For example, the performance requirements for a narrow latency range might be grouped together. If one stakeholder requests less than 2 seconds for transmitting an alarm while another requests less than 4 seconds, assuming low-end hardware, the engineers can require a low-end, small deployment of the system to support the alarm latency of less than 2 seconds. However, if another stakeholder re-

quests the far more demanding alarm latency of 0.5 second, the requirements might include a constraint, such as a high-end server, to ensure the short latency is implementable and acceptable for the targeted application situation.

Derive the NFRs for the Platform

NFRs derived for the platform from stakeholder product NFR requests relate to the platform’s behavior as a whole. For example, portability and stability NFRs are usually defined with respect to the platform in all its application needs. Because stakeholder inputs are intended for products that are to be built on the platform, the derivation is usually necessary. A simplified stakeholder input might be “A product shall provide 99.98 percent availability.” Developed into a platform requirement, it becomes “The platform shall provide 99.99 percent availability.” This example may appear naive. However, when no architecture design exists, such a requirement is a starting point.

Our recent research proposes detailed NFR-derivation techniques,⁴ which we based on goal modeling.⁶ In addition, during the platform architectural evaluation, software architects might use techniques such as the architecture tradeoff analysis method (ATAM) to ensure that the architecture can support the requirement.⁷

Derive NFRs for the Services

Deriving NFRs for services based on stakeholder product NFR requests is similar to specifying NFRs for the components within a nonplatform software product. The key difference is that the service NFRs might have to directly address the stakeholders’ inputs for services as well as the platform NFRs.

For example, a service startup time must be consistent with the platform startup time while satisfying the stakeholders’ service start-up requests. To

RELATED WORK IN DEVELOPING PLATFORM NFRS

Several software engineering techniques are available to help develop platform nonfunctional requirements (NFRs).^{1–5} The Platform NFR Development (PND) process complements them in two ways.

First, PND provides a more complete way of integrating existing techniques for solving platform NFR challenges than the earlier prototype we suggested for understanding user requirements.¹ In addition, because PND must analyze terminologies from different application domains, it should make ontological techniques useful in unifying complex relations among terminologies.^{2,3}

The work of Juha Kuusela and Juha Savolainen is similar to ours in analyzing the common needs of different software applications—for example, their definition hierarchy for capturing and analyzing product line requirements,³ and their method of managing the consistencies among product line requirements.⁴ In contrast with PND, they capture variations of requirements for different products in a product line. PND focuses on identifying

common needs that a platform can effectively implement and different applications can reuse.

The PND process is also unique in its codification of a proven practice for eliciting, reconciling, and specifying NFRs for software platforms. PND reflects real industry experience.

References

1. X. Song et al., “S-RaP: A Concurrent Prototyping Process for Refining Workflow-Oriented Requirements,” *Proc. 13th IEEE Int’l Conf. Requirements Eng.* (RE 05), IEEE CS, 2005, pp. 416–420.
2. K.K. Breitman and J.C.S. do Prado Leite, “Ontology as a Requirements Engineering Product,” *Proc. 11th IEEE Int’l Conf. Requirements Eng.* (RE 03), IEEE CS, 2003, pp. 309–319.
3. N. Nan and S. Easterbrook, “So, You Think You Know Others’ Goals? A Repertory Grid Study,” *IEEE Software*, vol. 24, no. 2, 2007, pp. 53–61.
4. J. Kuusela and J. Savolainen, “Requirement Engineering for Product Families,” *Proc. 22nd Int’l Conf. Software Eng.* (ICSE 00), ACM, 2000, pp. 60–68.
5. J. Savolainen and J. Kuusela, “Consistency Management of Product Line Requirements,” *Proc. 5th IEEE Int’l Symp. Requirements Eng.* (RE 01), IEEE CS, 2001, pp. 40–47.

deal with this issue, a spreadsheet can be developed to manage the relational consistencies: the sum of the start-up time of the services should be less than the platform startup time.

Check for Testability

Because platform software aims to satisfy a variety of applications, NFRs often lack enough specificity to be testable.

In addition to the testing team’s reviews, the NFR specifications are examined by engineers for sufficient information to quantifiably specify the test procedure, including the testing environment. Requirement engineers should integrate this activity with feature release management to focus on the features that are to be released soon. That is, for the upcoming release, the relevant NFRs must be clearly testable.

Complete the Constraints

The “check for testability” activity provides inputs for completing the con-

straints, because it helps uncover the conditions under which tests should be performed.

For example, a necessary precondition of testing the platform’s start-up latency is that the operating system has started. Specifying this condition lets testers verify whether the platform fulfills the latency requirement.

Adjust the NFRs for Feasibility

Software architects must ensure that the implementation technologies will likely satisfy the NFRs. They can estimate service performance by analyzing results from prototype and component testing in the performance model.

If the analysis shows that the NFRs might not be achievable, requirements engineers might modify the constraints as part of the NFRs.

Lessons Learned

The project architects and testing teams understood and used our NFR

specifications without needing much further clarification. Before PND, the design and testing teams had halted work, waiting for clarification of the requirements. After using PND, the testing team won a project award.

Here, we describe the most notable lessons learned from the project.

Accurate Stakeholder Inputs

Stakeholders joined our project at different times. The requirements elicitation questionnaires we sent to the latecomers included answers from earlier stakeholders as examples. We found that some stakeholders reused the existing answers when they were allegedly unsure about their needs. Because these inputs didn’t necessarily reflect actual needs, they made our analysis of the stakeholder priorities difficult.

This situation could occur in eliciting any product or functional requirements. However, for platform development, stakeholders often are vague about their

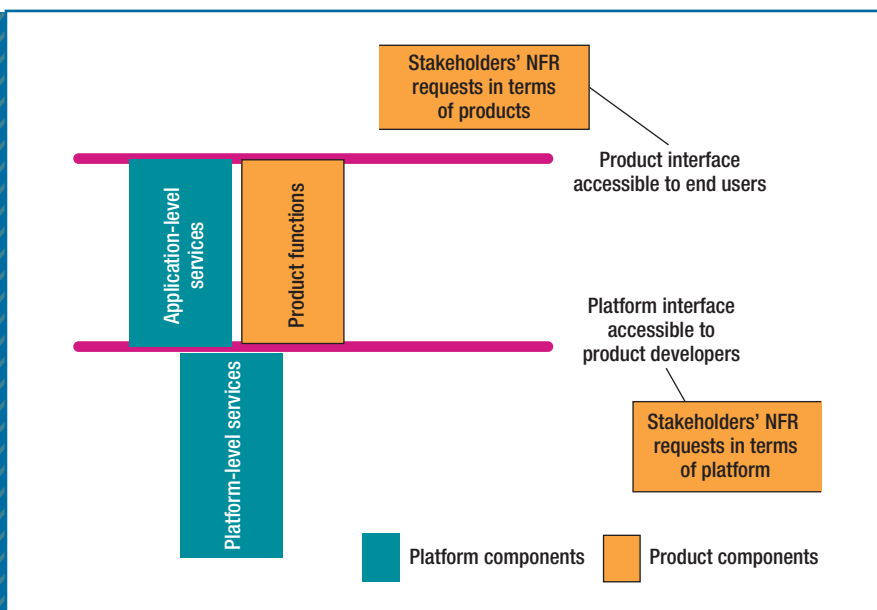


FIGURE 2. Application- and platform-level services. End users can directly access application-level services.

NFR needs or the exact NFR quantified value, so letting stakeholders see inputs from other stakeholders skews the results.

Lesson learned: keep each stakeholder from being influenced by other stakeholders' inputs to improve input accuracy.

Traceable Normalization and Reconciliation

Stakeholder inputs must be normalized before they're reconciled. Our experience indicates that stakeholders usually accept normalized inputs as long as the reconciled result is a more stringent requirement. However, we needed a clear trace to the original request to answer questions about the normalized inputs, especially when stakeholders reviewed the NFRs.

During the reviews, we used the traces to explain the reconciliations and their rationales. Without the traces and documentation of the normalizing and reconciliation method, the requirement team couldn't recall the basis and calculations used to derive the NFR values.

Lesson learned: make normalization and reconciliation traceable.

Constraint Reconciliation

Our project invested a great deal of effort in reconciling the conditions for the NFRs. We stored the stakeholder inputs and calculated data in a few thousand spreadsheet cells. Among those, 10 percent were for load conditions and system deployment configurations. The inputs from one stakeholder often were a set collected within one Siemens business group that planned to use the platform. Reconciling these load and deployment conditions was essential for NFR specification. It helped eliminate vague terms such as "small" footprint or "peak" load conditions.

Lesson learned: reconciliation must address not only requirements but also constraints such as deployment configuration and load conditions.

Automating Normalization and Reconciliation

We used spreadsheets for normaliza-

tion and reconciliation in our project because they supported traces and automated computations. Furthermore, the spreadsheets helped identify and manage the value ranges represented in stakeholder inputs. For example, we normalized one stakeholder performance requirement, "Alarms processed per 10 seconds," to alarms processed per second (APS). Then, we collected the APS needs from all the stakeholders in a single spreadsheet. Because each stakeholder provides the APS range (low and high value) for different deployment configurations, the spreadsheet automatically calculated the combined range by simply applying the maximum function to the high ends and minimum function to the low ends. In most cases, such simple calculations can lead to acceptable ranges (for example, the combined range was still useful input for making design decisions). However, we always reviewed the combined range from the automated calculation and sometimes overrode it (for example, narrowing or splitting it) to make sure the combined range wasn't too broad. In these cases, we also checked with stakeholders to make sure the reduced range was still acceptable.

Though we could apply the method only to the quantifiable stakeholder inputs, we found that the most difficult task in developing platform NFRs is managing the consistencies among quantifiable NFRs, where even minor quantity changes can cause a series of recalculations (for example, for normalization and re-reconciliation). Semi-automated analysis greatly improved our productivity for both creating and managing the NFRs and their traces.

Lesson learned: automating the normalization and reconciliation of quantifiable NFRs significantly improves productivity and requirements quality.

Deriving Stakeholder Inputs

At the project's beginning, we expected stakeholder inputs to apply to

the platform and the services it provided. However, after collecting the inputs, we found that a large percentage (60 to 70 percent) were really about the products to be built on the platform.

We realized that it's unrealistic to ask stakeholders to provide inputs for the platform because they have neither experience nor prior quality data about it. What the stakeholders know best are data from their existing, future, and competitor products. We had to derive the platform NFRs from the stakeholders' inputs for the products.

Lesson learned: stakeholders are often unable to provide inputs for platform NFRs.

Managing NFR Consistency

Deriving the NFRs for services is similar to specifying NFRs for the components within a (nonplatform) software product. The key difference is that NFRs for platform services might also have to directly address stakeholder inputs because other products might use the services as well. For example, stakeholder inputs might state performance rates for low-level message processing. Requirements engineers must evaluate these inputs to ensure the NFRs can support the services at both the application level and the platform level (see Figure 2).

Managing consistencies among stakeholders' inputs and NFRs at different levels is difficult and error-prone without proper tool support. Using spreadsheets to link related NFR values, sometimes even to link to the related stakeholder inputs, greatly improved management of NFR consistency.

Lesson learned: consistency among NFRs for a platform and its services is more complex than it is for nonplatform software products.

Incremental Testability Improvement

Early in our project, many NFRs weren't testable because constraints

ABOUT THE AUTHORS



XIPING SONG is a senior consultant at Siemens Corporate Research. His research interests include requirements engineering, medical workflow, software design, software regulation and standards, and software prototyping. Song received a PhD in computer science from the University of California, Irvine. Contact him at xiping.song@siemens.com.



BEATRICE HWONG is a consultant at Siemens Corporate Research. Her research interests include requirements engineering application and assessment, systems and software processes, and agile and global development. Hwong received a PhD in computer engineering from Rutgers University. Contact her at beatrice.hwong@siemens.com.



JOHANNES ROS is a Siemens-certified senior architect and a software development manager at Siemens Industry. His research interests include developing the science behind architecture as a product of practices and decision making, especially for global development projects. Ros received a PhD in computer science from the University of Pittsburgh. Contact him at johannes.ros@siemens.com.

such as deployment conditions weren't specified—a major issue that hindered the testing team. Structured templates help ensure complete constraint specifications. You can attach predefined constraints (specific load conditions, deployment configuration, redundancy status, and so on) to each NFR. We tried this approach in our project, but our experience with defining NFRs for all possible constraint combinations was overkill. Some combinations would be unlikely for the intended applications (for example, a small deployment configuration with redundancy). In addition, testing such NFRs would be costly. More efficient was focusing on a few likely combinations needed for near future releases.

Lesson learned: incremental improvement of NFR testability is effective and productive.


The PND process let us develop a detailed, well-organized NFR specification that consistently associated performance specifications with deployment configurations and load conditions. The specifications covered the needs for the near-future platform release, helping the project progress.

Defining NFRs for an application-domain-crossing platform is critical to the success of its development because the NFRs can represent key characteristics that separate the different domains. For example, speed as an NFR separates the aerospace from the automobile domain. At one extreme, satisfying NFRs for a wide variety of stakeholders could result in a platform impractical to implement regardless of how similar the stakeholders' functional needs are. Identifying the NFR needs and scoping

them to ensure that the platform is still implementable with the same base technology is complex analytical work. Our experiences indicate that a systematic process with tool support is essential to cope with the complexity.

The PND process greatly mitigated the problems associated with normalizing and reconciling stakeholder inputs and managing their consistencies. These problems are similar to those for developing any software but are complicated for developing platforms by the greater variety of range values and consistency types to check. The derivative nature of platform NFRs makes traces to the stakeholders' inputs difficult to recover once they're lost. Tool support for collecting, reconciling, and deriving the NFRs is essential for improving development productivity and

making the NFRs complete, consistent, verifiable, and traceable.

Our experience is limited to developing NFRs for a platform, but our experiences apply to all software systems that have a wide variety of application domains. The term unification, scale normalization, and stakeholder request reconciliation are critical to the NFR specification of such systems. 

References

1. E. Johansson et al., "The Importance of Quality Requirements in Software Platform Development—A Survey." *Proc. 34th Hawaii Int'l Conf. System Science (HICSS 34)*, vol. 9, IEEE CS, 2001, p. 9057.
2. X. Song, B. Hwong, and J. Ros, "Experiences in Developing Quantifiable NFRs for the Service-Oriented Platform," *Proc. 17th IEEE Int'l Conf. Requirements Eng. (RE 05)*, IEEE CS, 2009, pp. 337–342.
3. IEEE Std 830-1998, *Recommended Practice for Software Requirements Specifications*, IEEE, 1998.
4. D. Gross, E. Yu, and X. Song, "Developing Non-Functional Requirements for a Service-Oriented Application Platform: A Goal and Scenario-Oriented Approach," *Non-Functional Properties in Service Oriented Architecture: Requirements, Models and Methods*, N. Milanovic, ed., IGI Global, 2011, pp. 24–47.
5. N. Nan and S. Easterbrook, "So, You Think You Know Others' Goals? A Repertory Grid Study," *IEEE Software*, vol. 24, no. 2, 2007, pp. 53–61.
6. J. Mylopoulos, L. Chung, and B. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach," *IEEE Trans. Software Eng.*, vol. 18, no. 6, 1992, pp. 483–497.
7. L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, Addison-Wesley, 2003.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

IEEE computer society

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.

MEMBERSHIP: Members receive the monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others interested in the computer field.

COMPUTER SOCIETY WEBSITE: www.computer.org

Next Board Meeting: 11–15 June, Seattle, Wash., USA

EXECUTIVE COMMITTEE

President: John W. Walz*

President-Elect: David Alan Grier;* **Past President:** Sorel Reisman;* **VP, Standards Activities:** Charlene (Chuck) Walrad;† **Secretary:** Andre Ivanov (2nd VP);* **VP, Educational Activities:** Elizabeth L. Burd;* **VP, Member & Geographic Activities:** Sattupathuv Sankaran;† **VP, Publications:** Tom M. Conte (1st VP);* **VP, Professional Activities:** Paul K. Joannou;* **VP, Technical & Conference Activities:** Paul R. Croll;† **Treasurer:** James W. Moore, CSDP;* **2011–2012 IEEE Division VIII Director:** Susan K. (Kathy) Land, CSDP;† **2012–2013 IEEE Division V Director:** James W. Moore, CSDP;† **2012 IEEE Division Director VIII Director-Elect:** Roger U. Fujii†

*voting member of the Board of Governors

†nonvoting member of the Board of Governors

BOARD OF GOVERNORS

Term Expiring 2012: Elizabeth L. Burd, Thomas M. Conte, Frank E. Ferrante, Jean-Luc Gaudiot, Paul K. Joannou, Luis Kun, James W. Moore, William (Bill) Pitts

Term Expiring 2013: Pierre Bourque, Dennis J. Frailey, Atsuhiko Goto, André Ivanov, Dejan S. Milojicic, Paolo Montuschi, Jane Chu Prey, Charlene (Chuck) Walrad

EXECUTIVE STAFF

Executive Director: Angela R. Burgess; **Associate Executive Director, Director, Governance:** Anne Marie Kelly; **Director, Finance & Accounting:** John Miller; **Director, Information Technology & Services:** Ray Kahn; **Director, Membership Development:** Violet S. Doan; **Director, Products & Services:** Evan Butterfield; **Director, Sales & Marketing:** Chris Jensen

COMPUTER SOCIETY OFFICES

Washington, D.C.: 2001 L St., Ste. 700, Washington, D.C. 20036-4928
Phone: +1 202 371 0101 • **Fax:** +1 202 728 9614

Email: hq.ofc@computer.org

Los Alamitos: 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: 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: Moshe Kam; **President-Elect:** Gordon W. Day; **Past President:** Pedro A. Ray; **Secretary:** Roger D. Pollard; **Treasurer:** Harold L. Flescher; **President, Standards Association Board of Governors:** Steven M. Mills; **VP, Educational Activities:** Tariq S. Durrani; **VP, Membership & Geographic Activities:** Howard E. Michel; **VP, Publication Services & Products:** David A. Hodges; **VP, Technical Activities:** Donna L. Hudson; **IEEE Division V Director:** James W. Moore, CSDP; **IEEE Division VIII Director:** Susan K. (Kathy) Land, CSDP; **President, IEEE-USA:** Ronald G. Jensen

revised 24 Jan. 2012

