A New Standard for Quality Requirements

Jørgen Bøegh, *Terma A/S*

A new standard on software quality requirements, ISO/IEC 25030, takes a systems perspective and suggests specifying requirements as measures and associated target values.

The ISO recently published ISO/IEC 25030, a new standard on software quality requirements[1] that complements the two IEEE Computer Society standards for software[2] and system[3] requirements. These three standards are important, given that properly identifying and specifying requirements are prime factors in determining a software project's success or failure.[4] Many companies, having realized this, are now better emphasizing requirements specification. Unfortunately, there's a tendency to focus on functional requirements rather than quality issues such as usability, maintainability, reliability, portability, and efficiency.

ISO/IEC 25030 can improve software quality by helping developers identify and specify quality requirements. Here, as an editor and a member of the ISO committee, I discuss some of the thoughts behind ISO/IEC 25030 and briefly summarize its main points.

Developing the standard

The ISO first decided that quality requirements deserve their own standard back in 2001. It then implemented the idea in connection with a planned revision and restructuring of two international standards—ISO/IEC 9126 (which presents a software quality model)[5] and ISO/IEC 14598 (which discusses software product evaluation).[6] The ISO/IEC JTC1 SC7 committee included the quality requirements standard in a new SQuaRE (*S*oftware Product *Qua*lity *R*equirements and *E*valuation) series of standards, designated by the number 25000.[7] SQuaRE includes five divisions: quality management, quality model, quality measurement, quality requirements, and quality evaluation (see table 1). The quality requirements division contains the new ISO/IEC 25030 standard.

Taking a systems view of quality

When writing the standard, the committee members quickly realized that you can't elicit software quality requirements without taking a systems perspective. Software is normally part of a larger system, so you must view software requirements as part of the system requirements. We therefore looked at ways to describe systems. We wanted to develop a system model that was powerful enough to capture software quality requirements yet was easy to understand and still focused on quality.

A system is a combination of interacting elements organized to achieve one or more stated purposes.[6] The simplest system of possible interest when considering software quality is a computer system, which comprises three elements: hardware, software (including the operating system and application software), and data. This system model covers software running on a single, stand-alone computer and has been the implicit conceptual model behind software quality for many years.

However, the computer system isn't a realistic model. Software is often distributed on many computer systems—consider, for example, client-server systems and Internet applications. We needed to

0740-7459/08/$25.00 © 2008 IEEE

Table 1

The SQuaRE (*S*oftware Product *Qua*lity *R*equirements and *E*valuation) series of standards

Standards	Division	Description
ISO/IEC 2500n	Quality management	These standards define all common models, terms, and definitions in the SQuaRE series. They guide users through SQuaRE documents using referring paths. They also include high-level practical suggestions to help users apply the proper standards to specific applications. The division also provides requirements and guidance for a supporting function that's responsible for managing software-product requirements specification and evaluation.
ISO/IEC 2501n	Quality model	This division includes a detailed quality model (based on ISO/IEC 9126) comprising characteristics for internal and external quality and for quality in use. Furthermore, the model decomposes the internal and external quality characteristics into subcharacteristics. This division also includes a data quality model.
ISO/IEC 2502n	Quality measurement	This division includes a software product quality measurement reference model, mathematical definitions of quality measures, and practical guidance for their application. Presented measures apply to internal and external quality and quality in use.
ISO/IEC 2503n	Quality requirements	ISO/IEC 25030, the only standard in this division, helps identify and specify quality requirements. Developers can use these quality requirements to elicit and define quality requirements for a software product to be developed or as input for an evaluation process.
ISO/IEC 2504n	Quality evaluation	These standards provide requirements, recommendations, and guidelines for software product evaluation, whether performed by independent third-party evaluators, acquirers, or developers (internally in the developing organization). It also presents support for documenting a measure as an evaluation module. This division is based on the ISO/IEC 14598 series of standards.

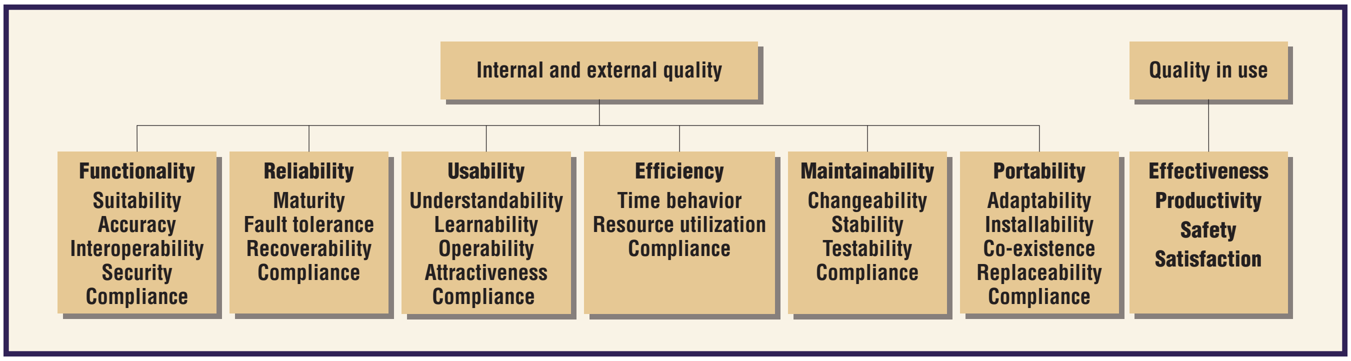

Figure 1. The ISO/IEC 9126 quality model.

model systems that comprise communicating computer systems, and we wanted to include embedded systems. So, we added a *mechanical parts* element that covers mechanics, electronics, hydraulics, and so on. This let us provide a system-description model covering a large class of applications.

But the real world is more complex, and not everything is automated, so we also added a *human-process* element. The system model includes communicating computer systems, mechanical parts, and human processes. This relatively simple hierarchical model of systems satisfied our need to describe software quality from the systems perspective.

Identifying a software quality model

In addition to the system model, we also needed a software quality model for describing quality requirements. The obvious choice was the ISO 9126 quality model,[5] which identifies a set of software quality characteristics and subcharacteristics (see figure 1). First published in 1991 and slightly enhanced in 2001, it's a well-known model that developers and researchers are using more and more in industry and in empirical research.

There are many opinions about what constitutes a software quality model's most important quality characteristics or factors and at what level these should appear in the model. Indeed, researchers have proposed several quality models with various sets of characteristics over the last 30 years. I doubt we've heard the last word on quality models. I think that the ISO 9126 quality model's most important advantage is that it's an international standard and thus provides an internationally accepted terminology for software quality. Whether we need to change some quality characteristics or subcharacteristics is of secondary importance in this context.

The ISO 9126 quality model presents three different views of quality. The first two, the internal and external views, share the same six characteristics and 26 subcharacteristics (see figure 1). The third view, quality in use, has its own four characteristics.

The internal view is concerned mainly with static properties of the software product's individual parts, including the design and code elements' structure and complexity. Developers can typically measure these quality properties early during development. A typical example of internal measures is Shyam Chidamber and Chris Kemerer's suite of "CK" measures for object-oriented software:[8] weighted methods per class (WMC), depth of the inheritance tree (DIT), number of children (NOC), coupling between object classes (CBO), response for a class (RFC), and lack of cohesion in methods (LCOM).

The external view is concerned with the completed software executing on the computer hardware, with real data. In this view, the software's dynamic aspects play an important role. A typical external measure is the mean time between failures, which relates to reliability in the quality model.

The quality-in-use view is concerned with the specified users performing specified tasks with the software in its real environment. This view typically measures end-user productivity and effectiveness.

These different views support each other. Internal quality influences external quality, which influences quality in use. Internal quality measures can act as early indicators for external quality. For example, if both the complexity of code (WMC) and the coupling between classes (CBO) are high, the software will likely be difficult to maintain. Similarly, external measures can indicate the quality in use—if the response time (efficiency) is low, end-user productivity will likely be low.

Internal-quality measures are also meaningful on their own. However, this isn't the case for external-quality measures, which depend on the computer hardware, the data, and possibly other elements. For example, an efficient algorithm implementation doesn't appear as an efficient program if the computer hardware is slow. When we consider quality in use, we might also need to consider mechanical parts and human processes. The quality of a system's individual parts plays a role in our conception of (software) quality. So, we have to consider the quality of the computer hardware, data, mechanical parts, human processes, and so on.

Identifying software measures. Before we can identify software quality requirements, we must clearly understand what the quality of a product really means. There are (at least) two different viewpoints:

- satisfaction of requirements (according to specifications) and
- satisfaction of stated and implied needs (fit for purpose).

These two viewpoints only coincide when the requirements specification reflects the stated and implied needs of all stakeholders for all applications. This is usually not the case. Many stakeholders can't articulate or don't even know their real needs. In addition, stakeholders might have conflicting needs. When we look at requirements from a software acquirer or supplier viewpoint, then their common interest is "satisfaction of requirements." The end users and public authorities (for example, in the case of safety-critical applications) are interested in "satisfaction of stated and implied needs." The first case focuses on the software product, while the second case focuses on the system.

Both viewpoints are important, so the standard takes both into account.

In the ISO quality model, a software quality (sub)characteristic is defined as a category of software quality attributes that influence software quality. An attribute is a measurable property—for example, size, which can be measured as the number of lines of code. We can determine a software product's behavior though its inherent properties and its quality through its inherent quality attributes. So, software measurement becomes the link between the quality model and the software's quality—that is, we can quantify software quality using software measures.

This implies that we can specify software quality requirements by providing a set of quality measures with associated target values. An example is an online administrative system with report-generation features. A possible quality requirement for this system is

- Quality characteristic: Efficiency (time behavior).
- Attribute: Report-generation time.
- Measure: Average number of seconds to generate report X during normal system use, measured 10 times using a stopwatch.

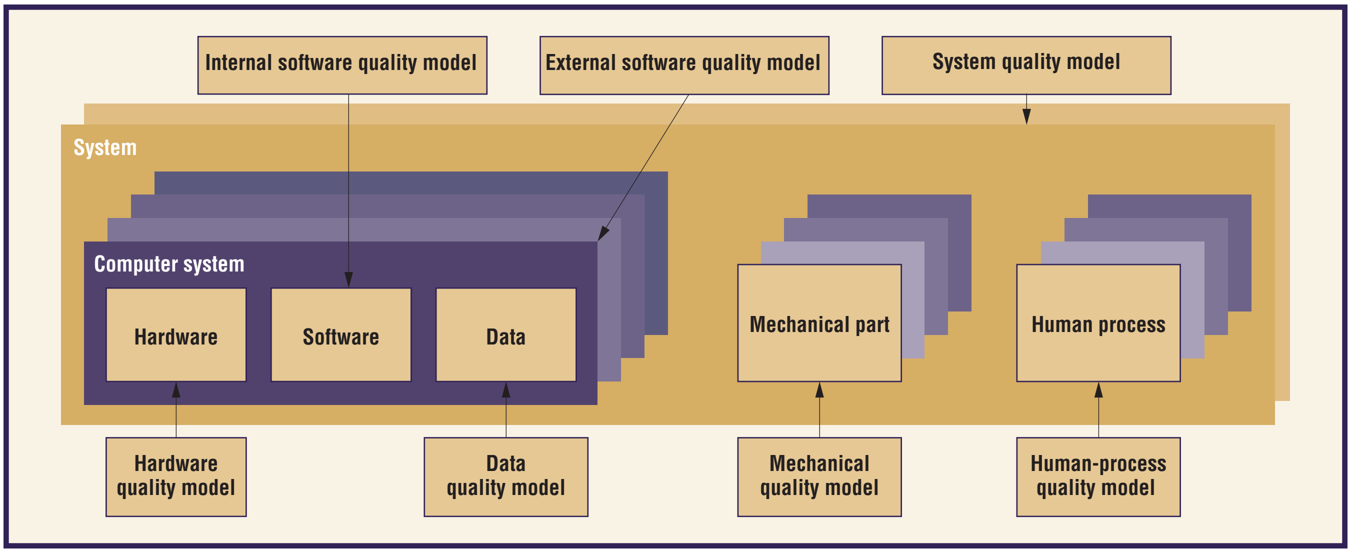

Figure 2. The quality models for the various elements of a system.

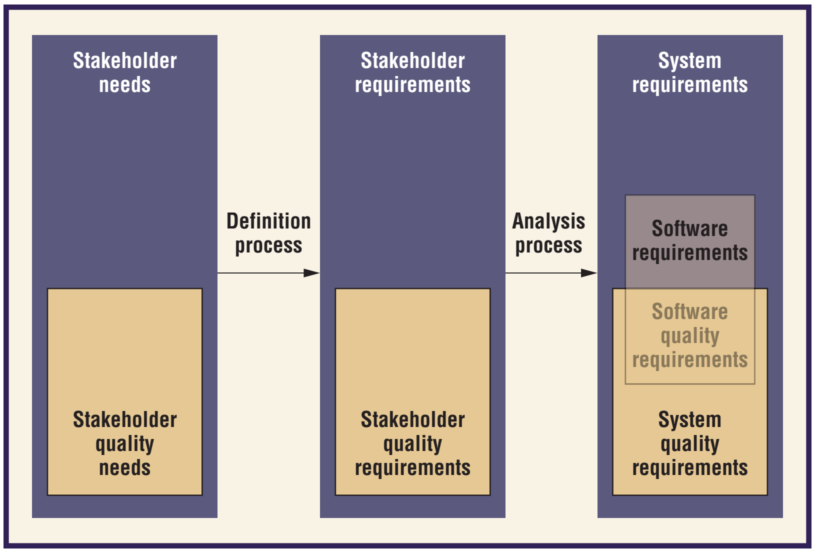

Figure 3. The software quality requirements definition and analysis processes.

- Target value: 20 seconds (60 seconds is the worst acceptable time).

When the software is completed, we can measure the quality attributes and compare actual measurement values to target values to reveal whether the software fulfills its quality requirements. It's important to formulate quality requirements such that we can demonstrate their fulfillment in a reasonable amount of time and with reasonable effort. What's "reasonable" depends on the stringency of the requirements and on the intended application. For a high-risk application, we're willing to spend more time and effort when evaluating quality.

Dealing with difficulties. Several difficulties arose in considering the ISO 9126 software quality model.

In particular, we weren't sure how to interpret functionality. Does it refer to the software's "functional abilities"? If yes, then the functional requirements would be a subset of the quality requirements. However, we instead decided to interpret all quality characteristics as statements about how well the software and its functions perform—that is, we focused on the functions' reliability, usability, efficiency, and so on. This becomes clearer when looking at functionality's five subcharacteristics (see figure 1):

- suitability (of the functions provided),
- accuracy (of the functions' results),
- interoperability (with other software),
- security (the ability to protect information), and
- compliance (the adherence to standards, laws, and regulations).

The model's size also made it complicated. Specifying quality requirements for a software product to this level of detail would be a major task. Fortunately, in most cases, this isn't necessary. For example, an aircraft engine is an embedded system with no direct end users, so we don't expect many usability requirements but instead stringent reliability requirements.

So, developers should use the quality model as a checklist to ensure they've included all important quality requirements. When specific quality needs exist, the developers should note them and specify associated quality requirements. It's equally important to avoid specifying too many or overly stringent quality requirements. This would be a waste of time and money, because fulfilling stringent software quality requirements is expensive.

1. Scope
2. Conformance
3. Normative references
4. Terms and definitions
5. Software quality requirements framework
    5.1. Purpose
    5.2. Software and systems
    5.3. Stakeholders and stakeholder requirements
    5.4. Stakeholder requirements and system requirements
    5.5. Software quality model
    5.6. Software properties
    5.7. Software quality measurement model
    5.8. Software quality requirements
    5.9. System requirements categorization
    5.10. Quality requirements life cycle model
6. Requirements for quality requirements
    6.1. General requirements and assumptions
    6.2. Stakeholder requirements
        6.2.1. System boundaries
        6.2.2. Stakeholder quality requirements
        6.2.3. Validation of stakeholder quality requirements
    6.3. Software requirements
        6.3.1. Software boundaries
        6.3.2. Software quality requirements
        6.3.3. Verification of software quality requirements

Annex A (Normative). Terms and definitions
Annex B (Informative). Processes from ISO/IEC 15288
Annex C (Informative). Bibliography

Figure 4. The table of contents of ISO/IEC 25030: Quality Requirements.

For example, again consider an aircraft engine's software. The software is highly critical, so its development and testing require substantial effort. Demonstrating fulfillment of the stringent reliability requirements also requires significant effort. It would be meaningless to specify a similarly stringent reliability requirement for the report-generating system I mentioned in the previous section and to allocate another large effort to demonstrate its reliability. My recommendation is to strive for the right level of quality—not too much or too little.

The ISO 9126 quality model relates primarily to the computer system. Only the quality-in-use characteristic applies to the whole system, and only from a rather narrow perspective. Figure 2 shows the quality models for the other system parts in addition to the ISO quality model. ISO/IEC 25030 points out the relations between various quality models. However, it's still a research issue to define and integrate the different quality models into a coherent system quality model.

Defining the requirements processes

When eliciting requirements, we must consider the whole system. Most stakeholders aren't interested in whether their needs are implemented in the software, in the hardware, in the mechanics, or as a manual process. Stakeholders simply want the system to satisfy their stated and implied needs. This is the "fit-for-purpose" viewpoint.

After developers have collected, analyzed, consolidated, and agreed on all the stakeholders' needs, they must apply a high-level architectural design process to determine what the software will implement. An analysis activity then identifies all software-related requirements. As explained earlier, the developers must formulate these requirements in terms of measures and target values. Figure 3 shows this two-step approach, which complies with the system life-cycle processes defined in ISO/IEC 15288.[9] This standard defines a set of processes categorized as either project, enterprise, or agreement processes. Technical processes constitute a subset of project processes, including

- the *requirements definition* process, which defines the requirements that can provide the services that users need in a defined environment; and
- the *requirements analysis* process, which transforms the stakeholder requirement-driven view into a technical view of a product.

The technical view of the software quality requirements is the "according to specification" view of the requirements—that is, you can measure and verify them objectively.

The main difference between ISO/IEC 15288 and ISO/IEC 25030 is that the first takes a process view while the latter focuses on the product—that is, on defining the quality requirements. This view is similar to the two IEEE requirements standards.[2,3] However, the IEEE standards focus primarily on functional requirements, although they also include quality aspects. For example, IEEE 830 specifically mentions performance (which isn't considered a quality feature), reliability, availability, security, maintainability, and portability.[2]

Introducing ISO/IEC 25030

International standards follow a specific ISO-defined template. Figure 4 shows ISO/IEC 25030's table of contents.

Clause 1 presents the scope and objectives. The standard applies to both acquirers and suppliers and provides requirements and recommendations for specifying quality requirements. It's particularly useful for

About the Author

Jørgen Bøegh is the Safety & Quality manager at Terma A/S. He's also head of the Danish delegation to ISO/IEC JTC1/SC7 and is editor of three international standards in software quality requirements and evaluation. His research interests include software quality modeling, requirements specification, and quality evaluation. He received his MSc in mathematics and computer science from the University of Aarhus. Contact him at Terma A/S, Vasekaer 12, DK-2730 Herlev; jorgen_boegh@yahoo.dk.

- specifications (including contractual agreements and calls for tender),
- planning (including for feasibility analyses),
- development (including early identification of potential quality problems), and
- evaluation (including objective assessment and certification of software product quality).

Clauses 5 and 6 present the standard's main body. Clause 5, which is more informative and doesn't include any requirements, describes quality concepts such as modeling and measurement and how these concepts relate to each other.

Clause 6 contains the normative requirements. Claiming conformance to the standard requires fulfilling these requirements. (The standard states the general conformance requirements separately in clause 2.) However, the standard's usefulness isn't restricted to situations requiring formal conformance; it's equally useful as a general guide for defining software quality requirements.

Clause 6.1 includes general assumptions. In particular, it states that the standard doesn't assume or require any specific software development model. In addition, it provides useful references to related standards, including ISO 9001,[10] which states that top management shall ensure that customer requirements are determined and met with the aim of enhancing customer satisfaction. ISO 9001 also goes into more detail about specifying customer requirements, including requirements not stated by the customer but necessary for the specified or intended use, as well as statutory and regulatory requirements related to the product. ISO/IEC 25030 further elaborates on these requirements.

Clause 6.2 provides requirements and recommendations for defining stakeholder requirements. First of all, the user of the standard must describe the system's intended purpose (or at least ensure that such a description exists). The clause emphasizes the need to identify all legitimate stakeholders and describe their roles. Stakeholders include end users, organizations—such as the acquirer and developer organizations—as well as statutory and regulatory bodies. Stakeholders might have conflicting interests, and their needs might even change during the system life cycle. The standard doesn't promote any specific elicitation methods or techniques but provides general applicable guidance, which can be used together with specific approaches. This clause takes a system view, because stakeholders aren't generally concerned with implementation. The standard emphasizes the need to document all stakeholder needs, wishes, expectations, and desires, even if they're conflicting, too ambitious, or completely unrealistic. This list must be prioritized and consolidated to an agreed and validated set of stakeholder requirements.

Clause 6.3 provides requirements and recommendations for software quality requirements. It assumes that developers make high-level architectural decisions about how to implement system requirements and about which parts to implement in software. So, it helps them identify stakeholder quality requirements relevant to the software. They then must formulate these requirements in terms of software measures with associated target values. This set becomes the technical formulation of the quality requirements.

Formalizing the software requirements in terms of measures and target values forces the involved parties to carefully consider which quality requirements are necessary. Developers can use the software quality requirements to monitor and control software quality during development as well as to evaluate the final product. The standard emphasizes having relevant stakeholders verify and formally agree on the list of quality requirements.

I hope that the quality requirements standard is well received in the software community and is used in industry as well as in research and education. Although we devoted much effort to preparing the standard, there's always room for improvement. I strongly encourage the standard's users to report their experiences to the ISO committee, either directly or through their national standards bodies. We welcome—and seriously consider—constructive comments. We can make progress in the standards area only through active dialogue between the standards' users and the standards committees.

Acknowledgments

I thank members of ISO/IEC JTC1 SC7/WG6 and particularly Kazuhiro Esaki for their diligent work on ISO/IEC 25030.

References

1. *ISO/IEC 25030:2007, Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)—Quality Requirements*, Int'l Organization for Standardization, 2007.
2. *IEEE Std. 830-1998, Recommended Practice for Software Requirements Specification*, IEEE Computer Society, 1998.
3. *IEEE Std. 1233, Guide for Developing System Requirements Specification*, IEEE Computer Society, 2002.
4. J. Verner, B. Kitchenham, and N. Cerpa, "Estimating Project Outcomes," *Proc. 20th Int'l Conf. Software & Systems Eng. and Their Applications*, 2007.
5. *ISO/IEC 9126-1:2001, Software Engineering—Product Quality—Part 1: Quality Model*, Int'l Organization for Standardization, 2001.
6. *ISO/IEC 14598-1:1999, Information Technology—Software Product Evaluation—Part 1: General Overview*, Int'l Organization for Standardization, 1999.
7. *ISO/IEC 25000:2005, Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)—Guide to SQuaRE*, Int'l Organization for Standardization, 2005.
8. S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Trans. Software Eng.*, vol. 20, no. 6, 1994, pp. 476–493.
9. *ISO/IEC 15288:2002, Information Technology—Life Cycle Management—System Life Cycle Processes*, Int'l Organization for Standardization, 2002.
10. *ISO 9001:2000, Quality Management Systems—Requirements*, Int'l Organization for Standardization, 2000.

ADVERTISER INDEX MARCH/APRIL 2008

Advertiser/Product	Page Number
Better Software 2008	**Cover 2**
ESRI	**9**
Java One 2008	**1**
Seapine Software, Inc.	**Cover 4**
STPCon 2008	**Cover 3**

***Boldface denotes advertisements in this issue.**

Advertising Personnel

Marion Delaney
IEEE Media,
Advertising Director
Phone: +1 415 863 4717
Email: md.ieeemedia@ieee.org

Marian Anderson
Advertising Coordinator
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: manderson@computer.org

Sandy Brown
IEEE Computer Society,
Business Development Manager
Phone: +1 714 821 8380
Fax: +1 714 821 4010
Email: sb.ieeemedia@ieee.org

Advertising Sales Representatives

Mid Atlantic (product/recruitment)
Dawn Becker
Phone: +1 732 772 0160
Fax: +1 732 772 0164
Email: db.ieeemedia@ieee.org

New England (product)
Jody Estabrook
Phone: +1 978 244 0192
Fax: +1 978 244 0103
Email: je.ieeemedia@ieee.org

New England (recruitment)
John Restchack
Phone: +1 212 419 7578
Fax: +1 212 419 7589
Email: j.restchack@ieee.org

Connecticut (product)
Stan Greenfield
Phone: +1 203 938 2418
Fax: +1 203 938 3211
Email: greenco@optonline.net

Southwest (product)
Steve Loerch
Phone: +1 847 498 4520
Fax: +1 847 498 5911
Email: steve@didierandbroderick.com

Northwest (product)
Lori Kehoe
Phone: +1 650 458 3051
Fax: +1 650 458 3052
Email: l.kehoe@ieee.org

Southern CA (product)
Marshall Rubin
Phone: +1 818 888 2407
Fax: +1 818 888 4907
Email: mr.ieeemedia@ieee.org

Northwest/Southern CA (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Midwest (product)
Dave Jones
Phone: +1 708 442 5633
Fax: +1 708 442 7620
Email: dj.ieeemedia@ieee.org

Will Hamilton
Phone: +1 269 381 2156
Fax: +1 269 381 2556
Email: wh.ieeemedia@ieee.org

Joe DiNardo
Phone: +1 440 248 2456
Fax: +1 440 248 2594
Email: jd.ieeemedia@ieee.org

Southeast (recruitment)
Thomas M. Flynn
Phone: +1 770 645 2944
Fax: +1 770 993 4423
Email: flynntom@mindspring.com

Midwest/Southwest (recruitment)
Darcy Giovingo
Phone: +1 847 498-4520
Fax: +1 847 498-5911
Email: dg.ieeemedia@ieee.org

Southeast (product)
Bill Holland
Phone: +1 770 435 6549
Fax: +1 770 435 0243
Email: hollandwfh@yahoo.com

Japan (recruitment)
Tim Matteson
Phone: +1 310 836 4064
Fax: +1 310 836 4067
Email: tm.ieeemedia@ieee.org

Europe (product)
Hilary Turnbull
Phone: +44 1875 825700
Fax: +44 1875 825701
Email: impress@impressmedia.com