Software Quality Requirements:
How to Balance Competing Priorities

J. David Blaine, *independent consultant*

Jane Cleland-Huang, *DePaul University*

Software quality is perhaps one of the most sought-after but unattained goals in software development. It's often understood, in a rather limited way, to mean the act of ensuring that a product meets its specified requirements. Unfortunately, this narrow viewpoint has led to the situation in which many quality assurance departments focus almost exclusively on testing and review and fail to address the more far-reaching questions that might differentiate between high-quality and mediocre renderings of the same product.

0740-7459/08/$25.00 © 2008 IEEE

Just for a moment, conjure up an image of a completely bug-free system that meets all stated requirements. Now ask yourself whether it delivers quality. The answer isn't as evident as it might first appear. The system clearly delivers all specified functionality, but the real question is, does it achieve this in a way that fully satisfies or perhaps even excites the stakeholders? What if the stakeholders failed to articulate their response-time expectations and the delivered product worked more slowly than anticipated? Or what if the system failed to appeal to the targeted user groups and therefore never made the necessary market breakthrough?

Both practitioners and researchers have widely recognized the need for a system to deliver real quality to its stakeholders. For example, Karl Wiegers identified several categories of quality requirements including attributes related to integrity, interoperability, performance, security, safety, usability, and testability.[1] As experienced practitioners know, this list is potentially quite long.

Terminology

Interest in system quality has led to a proliferation of related terms. The term *nonfunctional requirement* has been used broadly in academic literature and some parts of industry, but it suffers from an unfortunate ambiguity: "nonfunctional" is an actual word that, according to *Merriam-Webster's Collegiate Dictionary* (11th edition), means something doesn't work. Some people have adopted the term "ilities" in recognition that many of these qualities end in the suffix "ility." However, participants at a birds-of-a-feather session during the 14th International Conference on Requirements Engineering (RE 06) were unable to agree on a "one-size-fits-all" name for these kinds of requirements.

In response, Martin Glinz presented a vision paper at RE 07 in which he proposed a taxonomy of terms (see the sidebar "Classifying Requirements" on page 35 in this issue). For the purposes of this *IEEE Software* special issue, we've chosen to use the umbrella term *quality requirements*, even though we recognize that it introduces possible confusion between high-quality requirements and requirements that describe desired system qualities.

Challenges

In general, functional requirements specify *what* a system does, whereas quality requirements describe *how well* those functions are accomplished.[2,3] Unfortunately, quality requirements can be much more challenging to implement than functional ones for numerous reasons.

First of all, unless developers and analysts proactively elicit quality requirements, the stakeholders might well believe them to be implicitly understood. Obviously, this can lead to significant problems with stakeholder satisfaction of the delivered product.

Second, quality requirements tend to exhibit trade-offs that must be carefully negotiated and resolved.[4,5] For example, stakeholders might want a system to be both highly secure and easily accessible to users, or they might want a system to have very fast response times and support thousands of users but not cost much to build. Developers must find an architectural solution that balances these conflicting needs in a way that optimizes the delivered product's value.

Finally, quality requirements are often harder to measure and track than their functional counterparts. Whereas functional requirements are either present or not present in a system, quality requirements tend to be achieved at various levels along a continuum. This introduces challenges in their specification, tracking, and implementation.

Functional requirements specify what a system does. Quality requirements describe how well those functions are accomplished.

Choices, solutions

The benefits of getting the quality requirements "right" can be substantive, and in many safety-critical and real-time systems the developers have no choice but to deliver systems that meet very specific quality constraints. However, in most systems, we face a range of choices that will affect both the delivery schedule and a product's potential impact in the marketplace. We explore these choices in this issue of *IEEE Software*.

Following a rigorous review, we accepted five of the 25 submissions. These five articles provide well-balanced coverage of many of the issues and challenges related to our theme of balancing competing quality requirements.

The first article, "Making Practical Use of Quality Attribute Information," by Ipek Ozkaya, Len Bass, Raghvinder S. Sangwan, and Robert L. Nord, describes techniques for discussing quality attributes with stakeholders and for incorporating quality attributes into existing analysis and design methods. In the second article, "A Risk-Based, Value-Oriented Approach to Quality Requirements," Martin Glinz challenges established ISO standards by proposing a value-based approach for determining how to specify quality requirements. His article extends a panel presentation given at RE 07, held in November in Delhi, India.

The next two articles describe techniques for finding the right amount or balance of quality requirements. In "Supporting Roadmapping of Quality Requirements," Björn Regnell, Richard

About the Authors

J. David Blaine is an independent software quality improvement consultant specializing in project management and quality, with specific expertise in value planning, requirements engineering, and software measurement. He received an MA in mathematics from Arizona State University and an MS in electrical and computer engineering from the University of California, Santa Barbara. His professional accreditations include the Project Management Institute's Project Management Professional certification and the American Society for Quality's Certified Software Quality Engineer certification. Contact him at 7887 Embry Point, San Diego, CA 92126; jblaine@san.rr.com.

Jane Cleland-Huang is an assistant professor at DePaul University's College of Computing and Digital Media. Her research interests include requirements engineering with an emphasis on traceability and automated prioritization and triage. She coauthored *Software by Numbers: Low-Risk, High-Return Development* (Prentice Hall, 2003). She received her PhD in computer science from the University of Illinois at Chicago. Contact her at DePaul Univ., 243 S. Wabash Ave., Chicago, IL 60604; jhuang@cs.depaul.edu.

Berntsson Svensson, and Thomas Olsson introduce the QUPER (*qu*ality *per*formance) model. It helps stakeholders balance the costs and benefits of quality attributes in the face of time-to-market constraints, business opportunities, stakeholders' desires, and technical feasibility. Martin S. Feather, Steven L. Cornford, Kenneth A. Hicks, James D. Kiper, and Tim Menzies describe a defect detection and prevention model that supports key decision making concerning trade-offs between quality requirements. Their article is entitled "A Broad, Quantitative Model for Making Early Requirements Decisions."

Finally, in "A New Standard for Quality Requirements," Jørgen Bøegh explains the new ISO/IEC 25030 standard, which takes a systems perspective on software quality requirements. Bøegh's article describes some of the insights that went into this standard's creation.

In addition to these technical articles, Tom Gilb and Alistair Cockburn present their views in a lively Point-Counterpoint discussion on quantifying quality requirements. Tom sees the need to quantify each and every quality requirement, while Alistair's more agile perspective is that quality requirements should emerge during development.

We trust that the ideas and experiences reported in these articles will help you define, control, and achieve the levels of quality you desire in your projects.

Acknowledgments

We enthusiastically thank all the reviewers for their diligent attention and pragmatic suggestions in helping us define useful articles for *IEEE Software*'s readers.

References

1. K.E. Wiegers, *Software Requirements*, 2nd ed., Microsoft Press, 2003.
2. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
3. T. Gilb, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Butterworth-Heinemann, 2005.
4. L. Chung et al., *Nonfunctional Requirements in Software Engineering*, Kluwer Academic Publishers, 2000.
5. B.W. Boehm et al., "Using the WinWin Spiral Model: A Case Study," *Computer*, vol. 31, no. 7, 1998, pp. 33–44.

For more information on this or any other computing topic, please visit our Digital Library at www.computer.org/csdl.