IEEE Software
www.computer.org/software

A Risk-Based, Value-Oriented Approach to Quality Requirements

Martin Glinz

Vol. 25, No. 2
March/April 2008

A Risk-Based, Value-Oriented Approach to Quality Requirements

Martin Glinz, *University of Zurich*

This value-oriented approach to specifying quality requirements uses a range of potential representations chosen on the basis of assessing risk instead of quantifying everything.

When quality requirements are elicited from stakeholders, they're often stated qualitatively, such as "the response time must be fast" or "we need a highly available system." (See the "Defining Quality Requirements" sidebar for a definition of quality requirements.) However, qualitatively represented requirements are ambiguous and thus difficult to verify. As a consequence, we may encounter three kinds of problems:

1. The system developers build a system that delivers less than the stakeholders expect. This results in stakeholder dissatisfaction and might, in extreme cases, render a system useless.
2. The system developers build a system that delivers more than the stakeholders need. This results in systems that are more expensive than necessary.
3. The developers and the customer disagree whether the delivered system meets a given quality requirement—and there is no clear criterion to decide who is right.

For example, if the stakeholders mean 7 days × 24 hours of operation when they say "We need a highly available system" but the developers interpret this requirement as "at least 23 hours per working day," we have the first kind of problem. Conversely, if the stakeholders would be happy with availability from 6 a.m. to 8 p.m. on all work days while the developers build a 7×24 system with all the additional effort to develop and operate a continuously running system, we have the second kind of problem. Problems of type 1 typically also imply a problem of type 3.

The traditional way of solving these problems is to quantify all quality requirements. But quantification isn't the best solution in all cases. Instead, a quality requirement should be represented such that it delivers optimum value. You can determine such an optimal representation using a risk-based strategy.

Quantification

Quantification means defining metrics that make a requirement measurable (see the "Measuring Quality Requirements" sidebar). For example, we could quantify the requirement "The response time must be fast" as "The response time shall be less than 0.5 seconds in 98 percent of all user input actions." Work on quantification was pioneered by Barry Boehm[1] and Tom Gilb,[2] among others. Today, this topic is broadly covered by standards[3, 4] and textbooks.[5]

Some quality requirements are directly measurable—that is, a single well-defined metric adequately measures it. For example, performance requirements are directly measurable. The only difficulty in this case is to get the necessary quantitative input from the stakeholders—that is, motivat-

Defining Quality Requirements

The term *quality requirement* denotes those requirements that pertain to a system's attributes, such as performance attributes or specific qualities. For example, the following are quality requirements: "The system shall be user friendly," "The time interval between two consecutive scans of the temperature sensor shall be below two seconds," "The probability of successful, unauthorized intrusion into the database shall be smaller than 10^{-6}."

The term should not be confused with the notion of requirements that are of high quality—those that are adequate, unambiguous, consistent, verifiable, and so on.

There are different ways of positioning quality requirements in requirements classification frameworks. This article uses the classification shown in figure A, where quality requirements are denoted as *attributes*.[1] In this classification, system (or product) requirements are classified according to their concern. Requirements pertaining to a functional concern become functional requirements. A performance requirement pertains to a performance concern. A specific quality requirement pertains to a quality concern other than the quality of meeting the functional requirements. Eventually, a constraint is a requirement that constrains the solution space beyond what's necessary for meeting the given functional, performance, and specific quality requirements. Quality requirements, or attributes, are performance requirements or specific quality requirements.

The ISO/IEC 25030 standard classifies software product requirements according to the ISO quality model terminology (see figure B).[2] In this standard, software quality requirements ar a subcategory of *inherent property requirements*. The latter, together with *assigned property requirements*, forms the category of *software product requirements*.

References

1. M. Glinz, "On Non-Functional Requirements," *Proc. 15th IEEE Int'l Requirements Eng. Conf.* (RE'07), IEEE CS Press, pp. 21–26.
2. *ISO/IEC 25030: Software Engineering—Software Product Quality Requirements and Evaluation (SQUARE)—Quality Requirements*, Int'l Organization for Standardization, 2007.

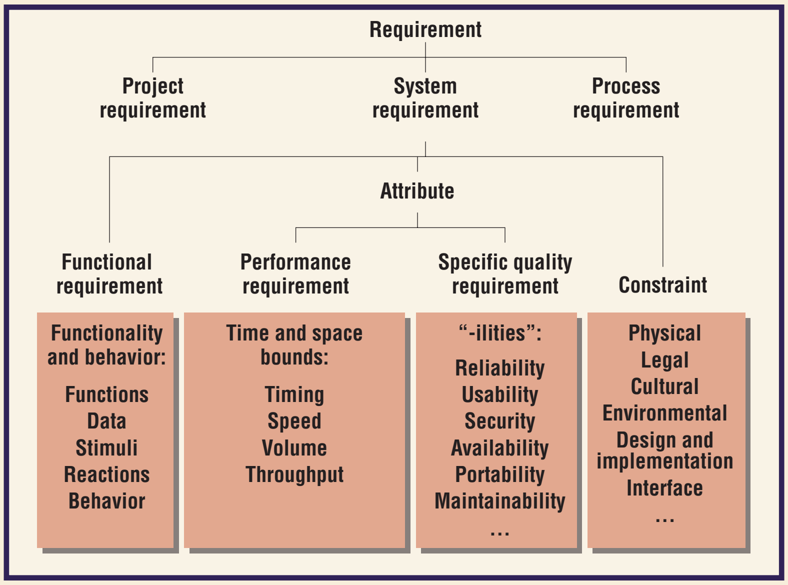


Figure A. The requirements classification used in this article.[1]

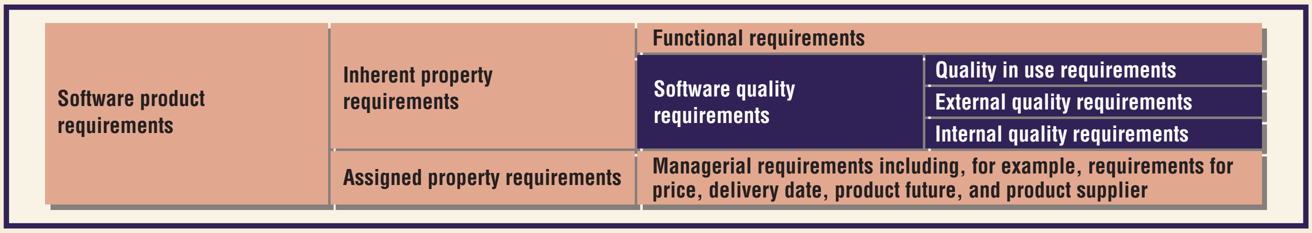

Software product requirements
Inherent property requirements
Functional requirements
Software quality requirements
Quality in use requirements
External quality requirements
Internal quality requirements
Assigned property requirements
Managerial requirements including, for example, requirements for price, delivery date, product future, and product supplier

Figure B. Classification of software product requirements according to ISO/IEC 25030.[2]

ing them to specify concrete threshold values such as "< 0.5 seconds" instead of "fast."

On the other hand, for some quality requirements, such a metric doesn't exist or its application is too expensive. Usability is a typical example of the first kind. We don't have a single metric for quantifying a requirement such as "The system shall be user friendly." Portability is an example of the second kind. We can measure it directly with the metric $M_{port}(s) = 1 - E_{port}(s) / E_{new}(s)$, where $E_{port}(s)$ is the average effort for porting the system s to a new platform and $E_{new}(s)$ is the average effort for developing s from scratch for a given platform. So the requirement "The system shall be highly portable" could be quantified as $M_{port}(s) \geq 0.8$. However, *calculating* this metric for a given system s would mean that $E_{port}(s)$ and $E_{new}(s)$ must be measured, which in turn would imply both porting s to the new platform and redeveloping s from scratch for the new platform (while keeping constant all other factors that influence the effort). Clearly, the cost of doing this is prohibitively high. Also, *estimating* $E_{port}(s)$ and $E_{new}(s)$

Measuring Quality Requirements

Measurement is the principle of making perceived attributes of an entity more objective by mapping attribute values to a *scale* such that the attribute's properties are mapped to corresponding properties of the scale. For example, if we have an attribute value a1 that is lower than another attribute value a2, the corresponding scale values s1 and s2 should be such that s1 < s2. A procedure for measuring an attribute together with a suitable scale is called a *metric*. Every scale has a type, which determines what we can do with the scale values. For example, on an *ordinal* scale, comparison is the only operation we can apply to scale values. In contrast, if we want to compute percentages and statistics such as mean and standard deviation, we need a *ratio* scale. More detail is beyond this article's scope. You'll find a comprehensive introduction to measurement and metrics in a textbook by Norman Fenton and Shari Lawrence Pfleeger.[1] The new ISO/IEC 25020 standard provides a reference model for software quality measurement.[2]

Defining a metric for a given attribute enables us to quantitatively assess attribute values—for example, comparing values or computing statistics. Intuitively, measuring a quality attribute requires at least a scale, a measurement procedure, a lowest acceptable value, and a planned value.[3,4]

For example, when we use this style, we can quantify the requirement "Need less time to service an incoming request than today" (see example of Jane's volunteer drivers' service in the text) as follows:

- Attribute: Average time that a dispatcher needs to service a request
- Scale: Seconds (type: ratio scale)
- Procedure: Measure time required from picking up a request to receiving the schedule confirmation from the system; take the average over 20 service requests. Web requests that can be scheduled automatically by the system count as zero.
- Planned value: 50 percent less than reference value
- Lowest acceptable value: 30 percent less than reference value
- Reference value: Average service request time as of today

References

1. N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed., PWS Publishing, 1998.
2. *ISO/IEC 25020: Software Engineering—Software Product Quality Requirements and Evaluation (SQUARE)—Measurement Reference Model and Guide*, Int'l Organization for Standardization, 2007.
3. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
4. T. Gilb, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*, Butterworth-Heinemann, 2005.

directly (without using subcharacteristics) is impossible except when an organization has ample experience both in porting and developing systems of similar size and complexity—and that's rare.

To measure these two kinds of quality requirements, we must find subcharacteristics of the given requirement that are directly measurable and whose values are strongly correlated with the values of the given quality requirement.

For example, the ISO/IEC 9126 standard[1-3] (see the sidebar "Standards for Quality Requirements") defines *usability* through the subcharacteristics understandability, learnability, operability, attractiveness, and usability compliance. Each of these again have directly measurable subcharacteristics. For example, a subcharacteristic of understandability is *completeness of description*, which we can measure as $C(s) = F_u(s) / F_{tot}(s)$ where $F_u(s)$ is the number of functions understood and F_{tot} is the total number of functions in a system s.

Practically speaking, the quantification of a quality requirement which is not directly measurable implies first defining an appropriate set of measurable subcharacteristics and then determining the required values for every subcharacteristic.

Advantages and drawbacks

The advantages of quantifying quality requirements are obvious: we get unambiguous, verifiable requirements and thus reduce the risk of delivering systems that don't satisfy stakeholders' desires and needs. So it's tempting to state "You shall quantify all quality requirements" as the first commandment of quality requirements engineering.

However, these advantages come with a price tag. For example, if we quantify the requirement "The system shall be user friendly" by using the usability characteristics given in the ISO/IEC 9126, we'd have to elicit required values for 28 directly measurable usability subcharacteristics and, when verifying the requirement, compute the actual values of 28 metrics. Worse, this might not even suffice. For example, in a Web-based order tracking system designed for use by untrained, casual users, an important usability subcharacteristic is the ratio of failed or aborted tracking attempts versus the total number of tracking operations. However, ISO/IEC 9126 doesn't include this characteristic as a subcharacteristic.

So, the disadvantage of quantifying quality requirements is equally obvious: it can be time-consuming and expensive. Dogmatically applying the rule "You shall quantify all quality requirements" might thus result in huge, unjustified requirement costs.

One might try to limit the cost of quantifying a quality requirement by quantifying only some selected subcharacteristics. However, focusing on achieving selected subcharacteristics and deliberately neglecting all others can easily result in a system that completely fails on the neglected ones, so that the total quality level of the system with respect to that requirement is lower than it could have been without any quantification.

A means, not an end

At this point, we should remember that requirements are a means, not an end. We want requirements that deliver *value*, defined here as the benefit of reducing development risk (developing a system that doesn't satisfy stakeholders' desires and needs) minus the cost of specifying the requirements.

Consequently, we should replace the rule "You shall quantify all quality requirements" with "A quality requirement should be represented such that it delivers optimum value." I'll demonstrate that risk assessment is the key means to determine how a given quality requirement should be represented.

This approach deviates both from classic thinking about quality requirements and from established quality standards such as ISO/IEC 9126[3] and ISO/IEC 25030,[4] where only a quantified quality requirement is a good quality requirement. However, reality is on my side: people have developed lots of successful software products without fully quantifying the quality requirements. Moreover, the new rule doesn't dismiss full quantification: it just provides a broader perspective, where full quantification is a valid option among others. By choosing the representation on the basis of a risk assessment, we also emphasize the fact that projects can fail if quality requirements aren't considered and treated adequately.

Risk-based analysis and representation

Basically, we must assess how every quality requirement should be represented so that it delivers the most value. This means assessing the risk of developing a system that doesn't satisfy the stakeholders' desires and needs with respect to a given quality requirement, and how we can mitigate this risk at the lowest possible cost.

A risk-based, value-oriented strategy for specifying quality requirements needs a broad range of representation forms, as characterized in table 1.

Whenever there's a high risk that the deployed system won't meet a quality requirement to the satisfaction of a critical stakeholder (and there's no way to weaken the requirement, lower the stakeholder's expectations, or shift the risk onto somebody else's shoulders), the best way to mitigate this risk is still a classic, comprehensive quantification of the requirement.

However, on the other end of the spectrum, if we have an implicit shared understanding among stakeholders and developers about a quality requirement, there's no need to specify it. The often-heard argument that not specifying implicit requirements is a recipe for disaster (and, hence, that implicit requirements always should be made explicit) doesn't pass a reality test. In our daily life, we have many contractual situations where we rely on an implicit shared understanding of requirements. For example, when a person orders a new car from a car dealer, she will not insist on putting requirements into the contract such as "The car shall be equipped with a working engine, passenger seats, and four tires pumped to the right pressure," because she relies on a shared implicit understanding of what comes with a new car. Problems with implicit requirements only arise when there's no or insufficient shared understanding. In this case, it's indeed important to make implicit requirements explicit.

Standards for Quality Requirements

ISO/IEC 9126[1–4] is the classic international standard for software product quality. In part 1 the standard defines a model of quality characteristics and subcharacteristics (see figure C). Parts 2–4 specify how to measure the subcharacteristics and define metrics for measuring them.

A new series of standards in the framework of SQuaRE (Software Quality Requirements and Evaluation) is under development. The important ones in this context are ISO/IEC 25010 (still being developed), which is planned to become the successor of ISO/IEC 9126; ISO/IEC 25020,[5] which specifies a measurement reference model; and ISO/IEC 25030,[6] which deals with quality requirements in software engineering.

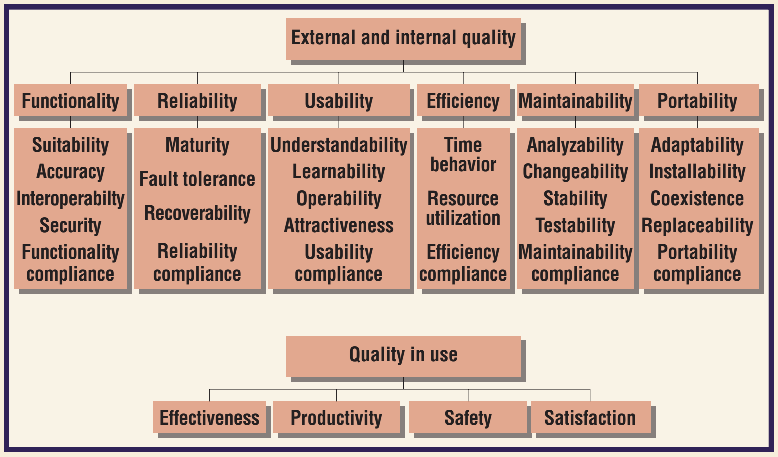


Figure C. Overview of the ISO/IEC 9126 quality model.

References

1. *ISO/IEC 9126-1: Software Engineering—Product Quality—Part 1: Quality Model*, Int'l Organization for Standardization, 2001.
2. *ISO/IEC TR 9126-2: Software Engineering—Product Quality—Part 2: External Metrics*, Int'l Organization for Standardization, 2003.
3. *ISO/IEC TR 9126-3: Software Engineering—Product Quality—Part 3: Internal Metrics*, Int'l Organization for Standardization, 2003.
4. *ISO/IEC TR 9126-4: Software Engineering—Product Quality—Part 4: Quality in Use Metrics*, Int'l Organization for Standardization, 2004.
5. *ISO/IEC 25020: Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)—Measurement Reference Model and Guide*, Int'l Organization for Standardization, 2007.
6. *ISO/IEC 25030: Software Engineering—Software Product Quality Requirements and Evaluation (SQuaRE)—Quality Requirements*, Int'l Organization for Standardization, 2007.

Table 1

The range of adequate quality requirements representation

Situation	Adequate representation	Verification technique
1. Implicit shared understanding among stakeholders and developers	Omission	Implicit
2. Need to state a general quality direction explicitly but trusting that the supplier will get the details right on this basis	Qualitative	Inspection
3. Involved parties have sufficient shared understanding to generalize properly from examples	By example	Inspection; may be some measurement
4. High risk of not meeting stakeholders' desires and needs adequately	Quantitative in full	Measurement
5. Somewhere between situations 2 and 4	Qualitative with partial quantification	Inspection and partial measurement

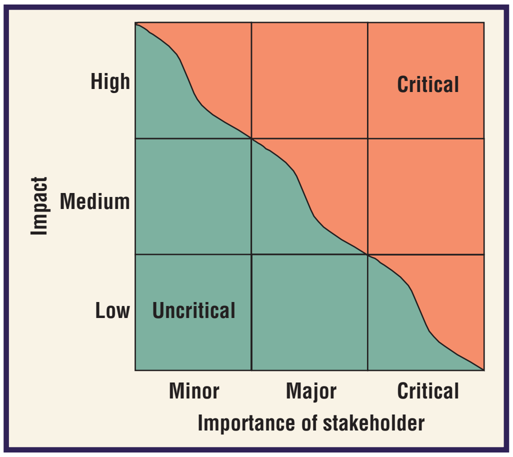


Figure 1. Assessing a quality requirement's criticality.

In situations where we have a degree of shared understanding that allows all involved parties to generalize properly from examples, specifying a quality requirement by giving an example can be very efficient. For example, you could specify a system's attractiveness by requiring it to be as attractive as a competitor's existing system.

When a manager assigns a task to an assistant in daily business life, she'll often give him only general direction, because she trusts him and knows that he'll carry out the task to her satisfaction. Any detailed task specification would be a waste of effort in this situation. An analogous situation can occur with quality requirements: for example, when a client has a history of successful collaboration with a supplier so that the supplier perfectly knows the client's business, problems, and needs, a qualitatively stated quality requirement such as "The new data entry function shall be recoverable" can suffice to achieve the value—that is, that the client is satisfied with the recovery features he gets for the new function.

The problem now is to assess, for a concrete quality requirement in hand, how to choose the representation that yields the most value.

One of the most important assessment criteria is the requirement's criticality (see figure 1). Along one dimension (the x-axis in figure 1), we analyze the importance of the stakeholder who states a quality requirement. For this purpose, stakeholders are classified in three categories: minor, major, and critical.[6] The other dimension (the y-axis) represents the impact of not meeting this requirement. Impact is typically regarded as the product of the damage incurred when a requirement isn't met and the probability that this will happen. The more critical a requirement is, the higher the probability is that quantification is necessary to achieve the best value. (Recall that "value" was defined earlier as the benefit of reducing the development risk minus the cost for specifying the requirements.)

Table 2 lists a set of factors that influence risk assessment. We might also use requirements triage techniques[7] to assess a quality requirement's relative importance.

Impact on elicitation and stakeholder analysis

Risk-based engineering of quality requirements, where we employ representations ranging from omission to full quantification, must not be misused as an excuse for treating quality requirements sloppily. In particular, quality requirements must be carefully and systematically elicited, no matter how they're eventually represented: you cannot assess quality requirements that haven't been elicited previously. The framework of characteristics and subcharacteristics provided by a quality model—for example, the one defined in ISO/IEC 9126—helps cover all areas of potential quality requirements when stakeholders don't state them spontaneously.

Equally important is a careful stakeholder analysis prior to eliciting requirements. Otherwise,

Table 2

Factors influencing risk assessment

Factor	Description
Criticality	The more critical a requirement, the higher the probability that quantification is necessary to achieve the best value.
Quantification effort	The lower the effort for quantification, the higher the probability that quantification will add value.
Validity of obtained measurements	When measuring a quality requirement indirectly through subcharacteristics, it is not a priori granted that the subcharacteristics' measurement values correlate with the degree of fulfillment of the requirement. The less evidence we have for these correlations, the less value we get from quantification.
Distinctiveness	When a quality requirement is distinctive for a product to be developed, not meeting such a requirement has a high impact on the product. So, a higher effort for specifying this requirement is justified than for nondistinctive requirements.
Shared understanding	When an implicit shared understanding exists about a quality requirement, specifying it explicitly has little value.
Reference systems	When a reference system (for example, a previous product or a competitor's product) exists, it might be much cheaper to specify a quality requirement by referring to the reference system's corresponding quality than by explicitly quantifying the requirement.
Length of feedback cycle	The shorter the feedback cycle (the time interval between expressing a need and receiving a system that should satisfy this need), the less verification and rework costs you can save through upfront requirements engineering. For quality requirements, this rule implies that the shorter the feedback cycle, the less payoff we get from quantifying the requirement.
Kind of customer-supplier relationship	The better the customer-supplier relationship (in terms of mutual respect and trust), the less requirements need to be contractually specified in an unambiguous and verifiable way—that is, the less we need quantified quality requirements.
Certification required	If an authority needs to certify a system, all certification-relevant quality requirements might need to be specified quantitatively.

the criticality assessment described in figure 1 is impossible.

Impact on verification

Verifying a quantified quality requirement is straightforward: after we've developed a system or component, we measure the property of the system or component that corresponds to the requirement and compare the obtained measure to the expected value(s) stored in the requirement. If we have precise and unambiguous metrics, the verification results will also be precise and unambiguous.

Giving up full quantification implies giving up this form of verification. The right-hand column of table 1 lists the verification techniques to be applied in different situations.

When a quality requirement is specified qualitatively, we need to substitute human judgment for measurement. This is not a priori bad and subjective: by using inspection techniques in the same way as in software inspections, we can objectify human judgment. Moreover, several metrics eventually also depend on human judgment. For example, the measurement procedure for the metric *Operational consistency in use* in ISO/IEC 9126 is "Observe the behavior of the user and ask the opinion."

If the specification of a quality requirement refers to an example, inspection is again the primary verification technique. In this case, it's also possible to measure and compare some core properties of the example and the developed system or component that pertain to the quality requirement to be verified.

A partially quantified quality requirement is verified by a combination of inspection and measurement.

If a quality requirement isn't specified owing to shared understanding, verification is implicit: as long as no stakeholder complains, the requirement is satisfied.

Example

Let's look at a hypothetical but realistic scenario to illustrate this approach.

The scenario: Jane's volunteer drivers' service

Jane's volunteer drivers' service is an organization of volunteers who drive elderly or disabled people when, for example, they must see a doctor, want to visit somebody, or wish to attend a religious service. When a person needs transportation, he calls the number of the volunteer drivers' service. Two dispatchers, Jane and Peter, alternate

Table 3

Analysis of quality requirements

Quality requirement	Stakeholder importance	Impact	Other considerations	Result
"Some weeks in advance"	Major	Low	Easy to quantify	Quantify. For example "Reservations can be made up to 21 days in advance."
"Very simple and easy to use"	Major	Low to medium*	Hard to quantify	Don't quantify. Reduce risk with user interface prototyping and let selected service users inspect the fulfillment of this requirement by working with the prototype.
"Same service level as provided by calling today"	Major	Low to medium	Reference system available, hard to quantify	Don't quantify. Instead, use current system as a reference system; maybe elaborate some examples that illustrate the current service level. Let selected service users inspect the fulfillment of this requirement by working with the prototype.
"Immediate confirmation in most cases"	Major	Low to medium	Easy to quantify	Quantify. For example, quantify "immediate" as "in less than 30 seconds" and "most cases" as "in at least 90% of all cases."
"Supports the growing number of service requests"	Critical	High	Other stakeholders also impacted, easily measurable, system architect must make a decision about upper limit for service requests anyway	Quantify. Specify values for the expected average number of service requests per week and the maximum number of service requests per week that the system shall be able to handle.
"As simple to use as our current spreadsheet"	Critical	Medium	Shared understanding between stakeholder and system architect, hard to quantify	Don't state explicitly as a requirement.
"Need less time to service an incoming request than today"	Critical	High	Distinctive, easy to quantify	Quantify (see example in "Measuring Quality Requirements" sidebar).

** People can still call instead of using the Web interface.*

in servicing incoming calls, making schedules, and calling volunteer drivers, giving them driving orders. They use a spreadsheet to create schedules.

Jane, with help from Peter and his wife, founded the service three years ago. Jane and Peter are friends and know each other's work habits and preferences pretty well.

Today, the service has grown to 20 volunteers and is still growing. The three founding members act as an executive board. With the organization's current size, the spreadsheet-based, manual dispatching no longer works efficiently. Therefore, the board has decided they need a computer system to support the ordering and dispatching processes.

In his professional life, Peter is the owner and chief engineer of a 10-person software company. He has informed his employees about this project and they've decided to contribute by building an open source, free system.

Stakeholder analysis

In this scenario, a stakeholder analysis would typically yield the following results: dispatcher would be *critical*; service user, volunteer driver, and system operator would be *major*; and developer, executive board member, person calling the service for somebody else, and any other stakeholder (for example, Internet access provider or hardware provider) would be *minor*.

Some requirements

Let's assume that stakeholders make the following statements (among others) when being interviewed. Lucy (a 76-year-old service user): "It would be great if I could make reservations over the Internet some weeks in advance and also view a list of my current reservations—I am a bit forgetful, you know. However, it must be very simple and easy to use." John (a deaf service user): "As I can't make calls myself, I need a Web-based reservation option. It should have the same service level as provided by calling today, in particular immediate confirmation in most cases." Jane (dispatcher): "My biggest concern is that the system must support the growing number of service requests but remain as simple to use as our current spreadsheet." Peter (the dispatcher and head of the development team): "I primarily want the system to help me work faster—that is, on average, I'll need less time to service an incoming request than today."

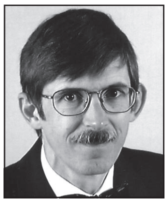
Risk-value assessment

General considerations: The customer and the supplier organization overlap in personnel (Peter, for example), and both organizations have a shared interest in producing a useful system. No certification or formal contract is required.

Table 3 presents an analysis of the individual quality requirements that we can extract from the requirements just stated.

Risk-based, value-oriented treatment of quality requirements extends the classic approach of making every quality requirement measurable. When we consider the value that the specification of a quality requirement adds, quantification is clearly not always the best way to represent a quality requirement. The approach shown here helps treat quality requirements adequately over a wide range of project situations and so helps advance software quality.

About the Author



Martin Glinz is a full professor of informatics at the University of Zurich. His interests include requirements and software engineering—in particular, modeling, validation, and quality—and software engineering education. He received his Dr. rer. nat. in computer science from RWTH Aachen University. Contact him at the Dept. of Informatics, Univ. of Zurich, Binzmühlestrasse 14, 8050 Zurich, Switzerland; glinz@ifi.uzh.ch.

References

1. B. Boehm, J.R. Brown, and M. Lipow, "Quantitative Evaluation of Software Quality," *Proc. 2nd Int'l Conf. Software Eng.*, ACM Press, 1976, pp. 592–605.
2. T. Gilb, *Principles of Software Engineering Management*, Addison-Wesley, 1988.
3. *ISO/IEC 9126-1: Software Engineering—Product Quality—Part 1: Quality Model*, Int'l Organization for Standardization, 2001.
4. *ISO/IEC 25030: Software Engineering—Software Product Quality Requirements and Evaluation (SQUARE)—Quality Requirements*, Int'l Organization for Standardization, 2007.
5. N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed., PWS Publishing, 1998.
6. M. Glinz and R. Wieringa, "Stakeholders in Requirements Engineering," guest editors' introduction, *IEEE Software*, vol. 24, no. 2, 2007, pp. 18–20.
7. A. Davis, *Just Enough Requirements Management*, Dorset House, 2005.