Effective Quality Management

Value- and Risk-Based Software Quality Management

Alexander Poth, Technical University of Berlin

Ali Sunyaev, University of Cologne

// Effective Quality Management combines value engineering and risk management to get acceptance of product quality assurance activities by developers, customers, and users. //

A KEY DELIVERABLE of software projects is adequate quality. Software quality has various definitions. The wider definition includes, besides other attributes, reusability and maintainability. Rosemary Stewart studied the restrictions of management in general and formulated a model dealing with demands, constraints, and choices.[1] In the context of quality management, we can refine her findings to two activities:

- delivering a product with the demanded quality, and
- making choices within the given constraints.

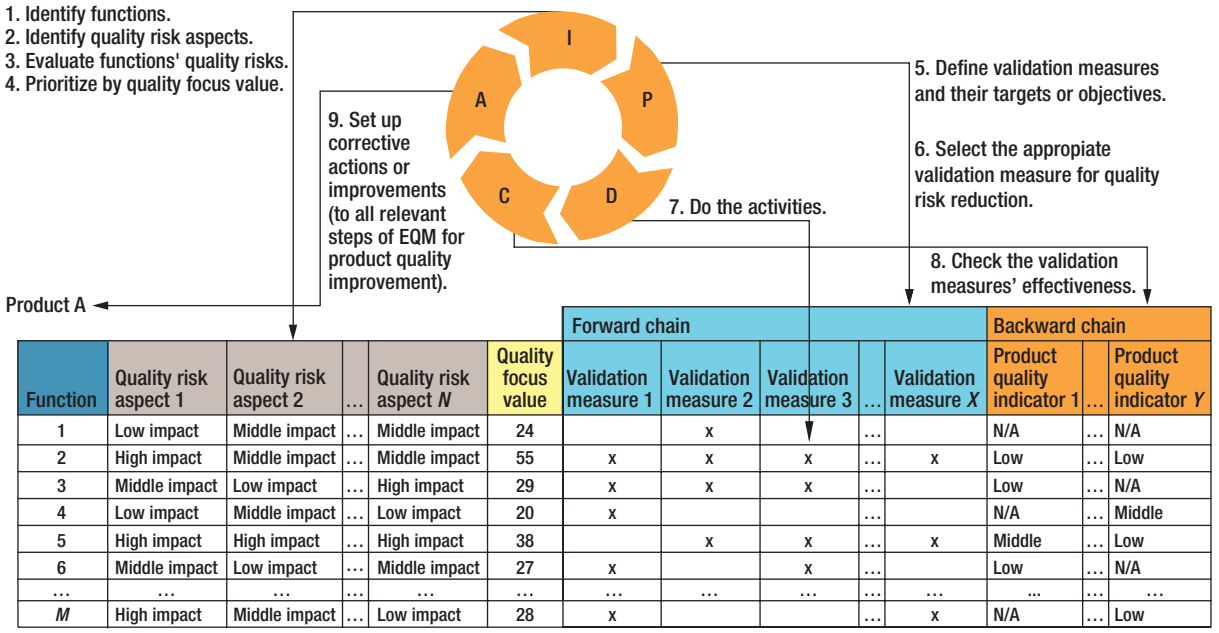
Considering companies' objectives in implementing adequate software quality, this article focuses on transparent, effective resource allocation of quality assurance (QA) activities for product verification and validation.

The method we present here resulted from our attempt to understand how quality managers and other software product stakeholders can come to an agreement about QA activities when making decisions about software functionality or features. We particularly wanted to improve the customer's acceptance of software quality at release time by using a structured method employing selected QA activities. At that point, the QA activities' transparency and results must give confidence to all parties in order for the software to be released. To give confidence in the delivery results, we developed our method systematically[2] to make transparent the tradeoff between mitigation of the quality risks and the constraint of limited resources for QA activities.

Recently, research on selected software development phases has received much attention as it has attempted to understand and improve software quality management (SQM) choices during the software life cycle. For example, James Bach proposed risk-based testing for the end of the development phase,[3] whereas Barry Boehm and LiGuo Huang suggested value-based software engineering as a holistic approach for software development that includes QA.[4] However, most research considers a non-holistic SQM viewpoint and focuses on development-phase-specific QA methods and activities.

To support SQM, we developed Effective Quality Management (EQM) to get acceptance of adequate product QA activities by a minimum set of stakeholders: developers, customers, and users. EQM employs constructive, analytic QA activities selected for the specific software product context. We've applied EQM to projects in domains from enterprise software to embedded software

0740-7459/14/$31.00 © 2014 IEEE



						Forward chain					Backward chain		
Function	Quality risk aspect 1	Quality risk aspect 2	…	Quality risk aspect *N*	Quality focus value	Validation measure 1	Validation measure 2	Validation measure 3	…	Validation measure *X*	Product quality indicator 1	…	Product quality indicator *Y*
1	Low impact	Middle impact	…	Middle impact	24		x		…		N/A	…	N/A
2	High impact	Middle impact	…	Middle impact	55	x	x	x	…	x	Low	…	Low
3	Middle impact	Low impact	…	High impact	29	x	x	x	…		Low	…	N/A
4	Low impact	Middle impact	…	Low impact	20	x			…		N/A	…	Middle
5	High impact	High impact	…	High impact	38		x	x	…	x	Middle	…	Low
6	Middle impact	Low impact	…	Middle impact	27	x		x	…		Low	…	N/A
…	…	…	…	…	…	…	…	…	…	…	…	…	…
M	High impact	Middle impact	…	Low impact	28	x			…	x	N/A	…	Low

FIGURE 1. Mapping the Identification-Plan-Do-Check-Act (IPDCA) cycle to the Effective Quality Management (EQM) matrix. The matrix implements all activities of the IPDCA cycle for application in software projects.

to prove this generic, holistic quality management method in practice.

EQM

EQM is based on value engineering (VE)[5] and risk management (RM),[6] motivated by Henry Gough's observation of VE and RM as "twin powers."[7] VE effectively identifies the value of a product's functions. General Electric developed the basic method after World War II to use restricted resources to implement the most valuable functions. VE is now integral to many engineering disciplines focusing on cost optimization. RM effectively identifies product risks and is a part of many industrial sectors' processes.

EQM assumes that variation exists in every software product's life cycle. The unexpected variation causes SQM adjustments to handle new situations with suitable quality assurance. EQM is based on the Deming cycle, also called the Plan-Do-Check-Act cycle,[8] an established quality process for continuous improvement.

EQM extends the Deming cycle with the Identification phase, which results in the Identification-Plan-Do-Check-Act (IPDCA) cycle (see Figure 1):

- The Identification phase identifies the product functions and systematically analyzes them to identify their value and risks, especially their quality risks.
- The Plan phase produces a set of *validation measures* (VMs), which are selected for suitable function validation to mitigate the functions' quality risks.
- The Do phase executes the VMs.
- The Check phase determines the VMs' effectiveness by checking *product quality indicators* (PQIs).
- The Act phase sets up corrective actions to improve product quality if needed. These actions can include adaptation of *quality-risk aspects* (QRAs), to improve the *quality focus value* (QFV); VMs (if new methods are available); or PQIs (to better reflect product use).

A row of the EQM matrix in Figure 1 contains a product function, such as "login to system," and its QRA evaluation, such as "workaround options and efforts" or "complexity of the function." The evaluation result is the QFV, which is mitigated by the selection of VMs such as "requirements review" or "unit tests." The function validation's effectiveness is monitored by

the values of the PQIs such as "number of defects." The evaluation and monitoring can use predefined value groups such as "middle impact" for easier evaluation or checks.

Although many industrial sectors have adopted VE and RM, and many publications have reported their successful application,[4,9] their combined use in engineering is still a matter of debate.[7] One major issue is how values and risks fit together. Despite this issue, we designed EQM so that value can be identified from the user's viewpoint. Risks should be identified from the user's and product developer's viewpoints. From the product developer's viewpoint, this entails the risk that a user's expected value won't be fulfilled. This leads us to express value as a QRA in EQM because EQM supports a product view of SQM during product development and maintenance.

EQM during the Product Life Cycle

We designed EQM to help SQM negotiate acceptable quality targets with the stakeholders and to adjust them over the product life cycle if needed. Often, some stakeholders, such as users or customers, don't personally participate in QA planning; they only review the QA strategy and plan. In these cases, SQM must compensate for the missing stakeholders during the QA planning meetings. Then, SQM must legitimize the plan for the stakeholders to accept. The same thing happens if the QA activities must adapt to unexpected occurrences that necessitate adjusting the planning.

EQM's IPDCA cycle guides SQM during the product life cycle.

Identification

The stakeholders' common terminology is the set of software functions and features. We use the function or feature list to break down the product into manageable units that are available over the product life cycle.

Furthermore, the stakeholders identify the product's value and QRAs. They can apply value by ranking the functions in a more VE style or in QRAs such as "mission relevance," "the percentage of function use per day," or "do-nothing costs." The QRAs are categorized into groups such as "high impact," "middle impact," and "low impact."

Then, the stakeholders evaluate the functions against the QRAs. If possible, all the relevant stakeholders participate in the evaluation, producing a common understanding of the function's values and risks. On the basis of the evaluation's results, they calculate the QFV:

$$QFV = \sum_{QRA=1}^{n} Value_{QRA}.$$

The stakeholders can use a *weighting factor* (WF) to prioritize QRAs and emphasize them in different phases of the product life cycle:

$$QFV = \sum_{QRA=1}^{n} \left(Value_{QRA} * WF_{QRA} \right).$$

Plan

In the Plan phase, the stakeholders first define the set of VMs for the functions. (The VMs handle verification activities, too.) VMs contain constructive, analytic QA activities such as reviews, static code analysis, and unit tests; if necessary, they also contain the coverage targets. If an established set of VMs is used, their costs and the effects in the product domain are known. This knowledge can help optimize QA's value and effectiveness through VM selection.

> We designed EQM to help software quality management negotiate acceptable quality targets with the stakeholders.

The stakeholders then define the VMs' targets, keeping in mind Tom DeMarco's observation that qualitative targets can be more useful in practice than quantitative targets.[10]

Depending on the demanded quality, the stakeholders can apply different best-practice strategies for assigning VMs to functions. For low-cost product development, we suggest starting with the function with the highest QFV and then deriving explicitly the minimum set of VMs, to mitigate the risks as much as necessary with minimal cost. This process proceeds until the budget is spent or all functions have a minimum set of VMs (the product costs are minimal).

If the budget has been spent before all the functions have VMs, the stakeholders must decide whether to increase the budget or live with the product's transparent validation gap. If the budget hasn't been spent, they should perform additional iterations to derive additional VMs. The iterations should continue until the budget is spent (the product quality fits the cost target).

As an alternative strategy, we suggest deriving VMs motivated by

Transmission controller A		Weighting factor							Forward chain				Backward chain		
		1.50	1.50	0.75	0.75		1								
Function	Feature	Criticality	Mission relevance	Internal complexity	External complexity	…	Expected change rate	Quality focus value	Review of system requirements	Static code analysis according to MISRA	Unit tests with >95% C1 coverage	…	100% traceability from requirements to test	Open change requests	…	Open bugs
Gear change	Neutral	Low impact	Middle impact	Middle impact	Middle impact	…	Middle impact	24		Mr. C, CW28		…	Mr. C, CW29	0	…	0
	Forward	High impact	Middle impact	High impact	High impact	…	Middle impact	55	Mr. A, CW24	Mr. C, CW28	Mr. A, CW29	…		2	…	1
	Backward	Middle impact	Low impact	High impact	High impact	…	High impact	29		Mr. C, CW28	Mr. B, CW29	…		3	…	1
Gear mode	Automatic	Low impact	Middle impact	Low impact	Middle impact	…	Low impact	20		Mr. C, CW28		…	Mr. C, CW29	1	…	0
	Manual	High impact	High impact	High impact	High impact	…	High impact	38	Mr. B, CW24	Mr. C, CW28	Mr. B, CW29	…		5	…	1
Diagnostic	RPM	Middle impact	Middle impact	High impact	Low impact	…	Middle impact	27	Mr. A, CW25	Mr. C, CW28	Mr. A, CW29	…		2	…	0
	…	…	…	…	…	…	…	…	…	…	…	…	…	..	…	…
	Clutch	High impact	Middle impact	Middle impact	Middle impact	…	Low impact	28	Mr. C, CW25	Mr. C, CW28		…	Mr. C, CW29	0		2

FIGURE 2. EQM as a matrix, focusing on the V-Model 97 process model for developing IT systems. This leads to a project-independent set of quality-risk aspects (such as "criticality" and "complexity") and validation measures (VMs).

the QFV. This way, the set of suitable VMs defines the quality budget. For each function, the stakeholders derive a set of VMs to mitigate the quality risks to an acceptable level.

In both strategies, functions with a higher QFV receive more resources for validation. Furthermore, the stakeholders discuss and decide together about spending resources for VMs.

When unexpected occurrences and changes to the project affect the QA resources, the SQM sets up corrective action mostly by reallocating VMs to handle the impacts.

Do and Check

The Do phase executes the VMs.

The Check phase controls the VMs' execution and updates the PQIs. We suggest using some development phase internal PQIs for the early setup of corrective actions and some customer or user PQIs for final product quality information. The customer or user PQIs—for example, bug reports—are important. Even though the internal PQIs might appear to produce good results, if the VMs are unsuitable or insufficient, quality issues might go undetected.

The stakeholders can define the PQIs during EQM setup. Examples are bugs from testing, customer or user feedback, and review findings, which are assigned to functions. To show the effectiveness of early error detection by VMs, a mapping of bugs, review findings, and so on to the internal PQIs is useful. To optimize the reaction to quality derivations, each development phase has its PQIs.

Act

The Act phase sets up a continuous-improvement process based on Kaizen.[11] If the PQIs don't meet expectations, the stakeholders perform a root-cause analysis, which typically leads to more knowledge about the selected VMs' effectiveness. The stakeholders use this knowledge to define the corrective actions and to monitor and improve the VMs' effectiveness.

Applying Our Method

We applied EQM in different product development contexts. Ultimately, we had positive experiences with stakeholder acceptance of EQM and its resource allocation for QA activities and adjustments during development.

The V-Model

This application of EQM involved an engineering company's development of electrical and electronic products. To evaluate functions and features, we used QRAs motivated mostly by the V-Model 97 process model (Development Standard for IT Systems of the Federal Republic of Germany—General Directive No. 250/3). Two such QRAs were "criticality" or "complexity." We added other QRAs such as "mission relevance," which expressed a function's value in the customer's or user's value view of the product (see Figure 2).

For each QRA, we evaluated and ranked the functions and features, such as the function "gear change" and the feature "neutral." We calculated the QFV on the basis of the quality-risk evaluation. The WF depended on the QRA's focus in the project or product lifecycle phase.

For some QRAs, we defined VM sets to fit V-Model 97 requirements, which associated the QRA evaluation results to VMs. VM sets define quality measures for a QRA's values. EQM defines VM sets that incorporate a group of independent VMs into a logical monolithic VM for mitigating a specific quality risk. We applied a VM set to the QRA "criticality." "Criticality" is independent of management choices because the associated VMs are constraints with predefined target values. One such VM is the

code coverage target for unit tests of critical functions.

To derive VMs that weren't predefined by VM sets, we used VMs based on a function's QFV and specific quality-risk profile. In our example, this was evident by our selecting more VMs for the "forward" feature than for the "neutral" feature to mitigate its higher QFV. Furthermore, we defined the VMs' quality target, such as Motor Industry Software Reliability Association (MISRA) conformance analysis, or the explicit unit-test-coverage value targets.

Next, we integrated EQM into the development process. This led to a function or feature list in the requirements specification tool that we could use for EQM. The project schedule and resource allocation (the person responsible for the VM) were defined in the project management tool, which handled all project tasks including QA tasks. We associated bugs and change requests with the function list, to have a backward chain that provides current quality information. We used the backward chain's information to validate the effectiveness of the functions' selected VMs. The direct mapping made it easier to classify bugs and change requests because we could use the QRAs as classification information. Furthermore, the association was useful for quickly estimating the validation effort because the function changes had to be validated according to EQM.

Scrum

This application of EQM involved the software for an airline's customer benefits program. The software involved developing two subsystems—one in India and the other in Germany. Both subsystems interacted with more than 50 other systems or interfaces. Owing to the different business cases or functions the overall system handled, we set up different teams for systems integration and user acceptance tests. Three functional teams were in different locations. A nine-member team dealt with the software for the benefits program's spending stage (in which customers spend their accumulated points); the team followed Scrum (scrumalliance.org).

First, we defined the QRAs for the software product context. Scrum (or agile project management) is value and priority driven. One way to map agile development's value paradigm to the QRAs is to define risk values in monetary terms. Instead, we took a qualitative approach. This required defining a set of comparable units (such as "low impact," "middle impact," and "high impact" in Figure 1). The project defined five types of value and risk. The context of the system's function was the spending environment at call centers, which dealt with different spending use cases and their activities. These activities constituted the software functions (*stories* in Scrum terminology).

The first QRA was "value" from the user's viewpoint. To make a function's contribution to the daily business transparent, "value" measured how often people used it (0–20, 21–40, 41–60, 61–80, and 81–100 percent).

We split the other function risks into two QRAs. One was based on the customer or user viewpoint of the probability of a loss-producing impact. (This QRA involved four situations: "no real impact," "impact with an inexpensive workaround that's hidden from the customer," "impact with an expensive workaround that's hidden from the customer," and "impact with a workaround that's visible to the customer and is a show stopper"). The other QRA made it transparent that some functions were important for a very small customer group that generated very high revenue. The quality risk is not to serve these valuable customers.

This set of QRAs covered all relevant value and risk drivers in the QFV. Our approach brought together value-driven agile development and risk-based quality management. We transformed values in a risk view, through an accepted table of classification units for values and risks. The point is that losing high-value functions because of defects incurs high costs, leading to quality risks that are high. Conversely, the quality issues due to these risks reduce a function's value.

> Through EQM's transparency, all projects have improved the relevant stakeholders' acceptance of their QA activities.

On the basis of this set of QRAs, prioritizing the functions' values was easy because adequate risk mitigation delivered more value per function. Furthermore, we defined a VM set for handling the quality risks. This led to a mandatory set and an optional set of QA activities (and selectively used measures such as "pair working versus reviews" or "early regression testing").

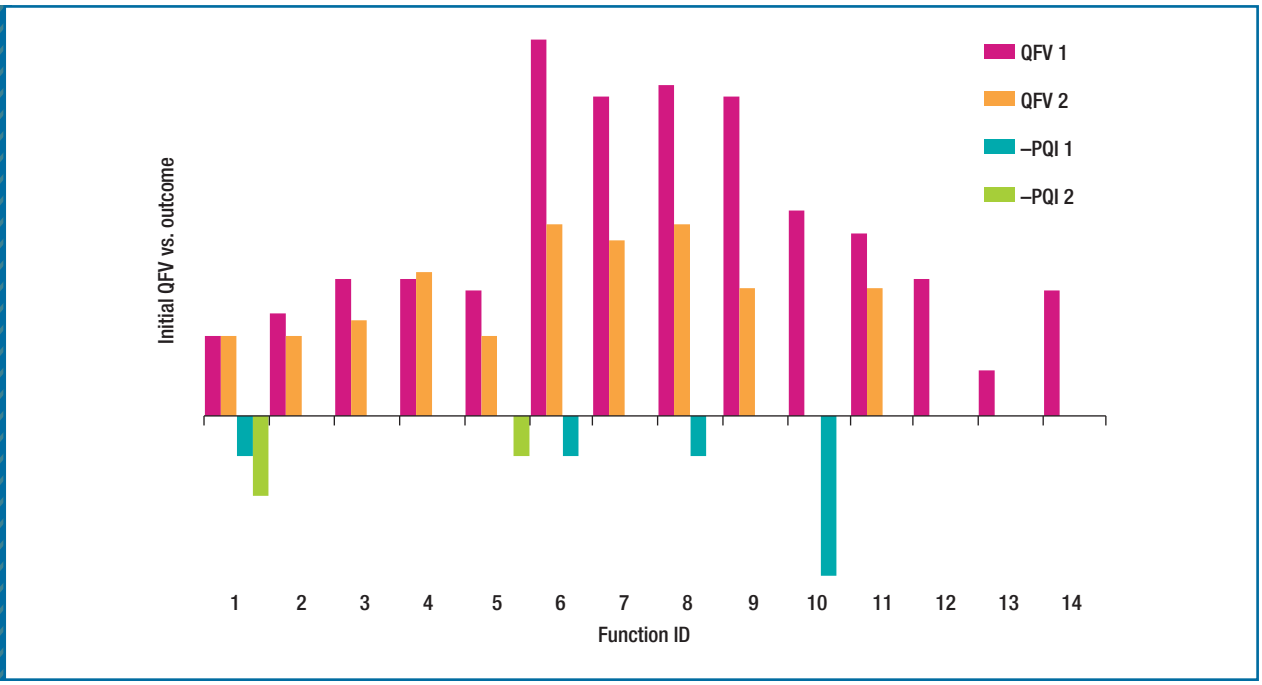


FIGURE 3. Two iterations of the IPDCA cycle with the evaluated quality focus values (QFVs) of product functions and the associated product quality indicator (PQI) "defects" after applying the derived mitigation VMs. During the iterations, the QFV became smaller and more precise. A smaller QFV led to fewer (cheaper) and more precise VMs, which led to better, more precise PQI values.

One PQI we used was "bugs per Scrum story or function."

SPICE

This application of EQM involved an automotive supplier's development of electrical and electronic products. To implement EQM in an organization's software product domains, we have to support tailoring of the EQM parameters. The challenge is to balance a specific definition of the parameters in a specific domain context with the definition's reusability for similar or other domain contexts.

EQM solves this issue by using both a flexible level of abstraction of the product by its functionality, which is applicable in almost all domains, and a flexible set of QRAs and PQIs. Furthermore, EQM fulfills normative requirements by its openness toward the chosen VMs.

SPICE (Software Process Improvement and Capability Determination; ISO/IEC 15504) requires associating VMs with processes. So, for a development environment to conform to SPICE, the applicable VMs must conform and align to the defined and applied process requirements of SPICE. We mapped the VMs to the processes of automotive SPICE, such as MISRA analysis, as a part of the unit verification strategy of SPICE's ENG.6 (software construction) process. ENG.6 is controlled by the SUP.1 (QA) process, which includes EQM as part of its project QA strategy.

Equally important for EQM implementation is the possibility of refining the EQM method parameters to allow step-by-step introduction of EQM. EQM provides this with an initial setup of functions, the QRA evaluation, and the VMs. In a next step, these could be refined or enhanced, once more information on the product becomes available and the organizational constraints or other details are provided. For example, after a few projects involving dynamic multinational development teams, we added an explicit QRA: "the developer's know-how or experience level."

For SPICE, an important activity for an organization is to train those employees who must be knowledgeable about EQM. Further constraints are change management and control of the implementation during a method rollout. After EQM rollout to the product development organization, we verified EQM's establishment through independent SPICE assessments

conducted by customer assessors in different international development locations.

We've designed, applied, and improved EQM over the past six years in different domains, in small and large projects, in product development programs, and in software systems development organizations. We've also gotten confirmations in SPICE assessments. Through EQM's transparency, all projects have improved the relevant stakeholders' acceptance of their QA activities. We believe EQM can help the persons or teams responsible for QA make the right choices and make product quality transparent, by monitoring quality risks and product evolution over the product's life cycle.

Figure 3 shows the life cycle of function development in a SPICE development context during two iterations (the *x*-axis) of the IPDCA cycle as relations of the QFV and PQI per function (the *y*-axis) and iteration. Functions with just one QFV are only included in one release. The figure shows that during the iterations, the QFV became smaller and more precise. A smaller QFV led to fewer (cheaper) and more precise VMs, which led to better, more precise PQI values.

EQM is useful for prioritizing QA efforts with a preventive focus. It's also useful for understanding the quality risks of functions and their changes and thus for prioritizing QA efforts effectively. For example, the QA team could prioritize efforts for new functions by selecting specific risk mitigation activities.

EQM should be used carefully when the quality risks are unclear and the effectiveness of risk mitigation by VMs is unknown. For example, if you're using only a default set of quality risks, the derived VMs won't always lead to good product quality. We can't recommend applying EQM without reflecting critically on the suitability of the identified quality risks and defined VMs.

To implement EQM in a running program or an organization, we recommend first preparing PQIs by finding the root causes of value and quality issues such as defects back from system operations or the IT service organization. Use the results to motivate the changes to EQM for SQM. Next, check whether all relevant VMs for a potential mitigation of the analyzed defects are available for the product developers. If not, start training people on VMs. In parallel with the VM analysis, you can define QRAs and evaluate the functions. The last step brings everything together in the IPDCA cycle.

ABOUT THE AUTHORS

ALEXANDER POTH is an IT quality manager for a major automotive original equipment manufacturer and was a senior consultant at Software Quality Systems AG. His interests are software engineering, quality management, and risk management. Poth received a Dipl.-Ing. (master's) in computer engineering from the Technical University of Berlin. Contact him at alexander.poth@alumni.tu-berlin.de.

ALI SUNYAEV is an assistant professor in the department of information systems at the University of Cologne. His research interests are information systems, security engineering, and health IT. Sunyaev received a PhD in computer science from Technische Universität München. Contact him at sunyaev@wiso.uni-koeln.de.

References

1. R. Stewart, *Choices for the Manager*, Prentice Hall, 1982.
2. A. Sunyaev, M. Hansen, and H. Krcmar, "Method Engineering: A Formal Description," *Information Systems Development: Towards a Service Provision Society*, Springer, 2010, pp. 645–654.
3. J. Bach, "Heuristic Risk-Based Testing," *Software Testing and Quality Eng. Magazine*, vol. 1, no. 6, 1999, pp. 22–29.
4. B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," *Computer*, vol. 36, no. 3, 2003, pp. 33–41.
5. S. Biffl et al., *Value-Based Software Engineering*, Springer, 2006.
6. D.W. Karolak, *Software Engineering Risk Management*, IEEE CS Press, 1995.
7. H.J. Gough, "Value and Risk—Twin Powers," *The Value Manager*, vol. 15, no. 1, 2009, p. 14.
8. W.A. Shewhart, *Statistical Method from the Viewpoint of Quality Control*, Dover, 1986.
9. Special Issue on Risk Management, *IEEE Software*, vol. 14, no. 3, 1997.
10. T. DeMarco, "Software Engineering: An Idea Whose Time Has Come and Gone?," *IEEE Software*, vol. 26, no. 4, 2009, pp. 95–96.
11. M. Imai, *Kaizen: The Key to Japan's Competitive Success*, Compañía Editorial Continental, 1991.