

How Much Software Quality Investment Is Enough: A Value-Based Approach

LiGuo Huang and Barry Boehm, *University of Southern California*

Based on the COCOMO II cost-estimation model and the COQUALMO quality-estimation model, quantitative risk analysis helps determine when to stop testing software and release the product.

Is quality really free, as Philip Crosby would have us believe?¹ Does a higher-quality Lexus cost less to produce and purchase than a lower-quality Corolla? Unfortunately, there are no simple, one-size-fits-all answers to such questions. Key stakeholders in a given project must analyze each quality investment situation to determine as well as possible what investment levels are not enough and what levels are too much.

A good example in the software field is how to determine when to stop testing and release the product for use. We've found that risk analysis helps address the question of how much testing by balancing the risk exposure (loss probability \times loss impact) of doing too little with the risk exposure of doing too much. However, people have often found it difficult to quantify the relative probabilities and economic value of loss to provide practical approaches for determining a risk-balanced "sweet spot" operating point.

This article draws on results from the emerging field of *value-based software engineering* (see the sidebar). VBSE aims to provide a quantitative approach to such questions as how much software quality investment is enough. In this case, the approach is based on the well-calibrated COCOMO II cost-estimation model² and the partially-calibrated COQUALMO quality-estimation model.³ We also provide examples of its use under differing value profiles that characterize early startups, routine business

operations, and high-finance operations. Further, we show how the model and approach can assess the relative payoff of value-based testing as compared to value-neutral testing.

Development cost of required reliability: Cocomo II

The core of the COCOMO II constructive cost model is a mathematical relationship involving 24 variables used to estimate the effort in person-months required to develop a software product. By multiplying the project effort by its cost per person-month, you can also estimate the project's cost.

COCOMO II's parameters include

- the product's equivalent size in thousands of lines of code (KLOC) or a function-point equivalent;
- personnel characteristics such as capability, experience, and continuity;

Value-Based Software Engineering

General frameworks for making software engineering decisions that enhance the value of delivered systems have been developed in the Economics-Driven Software Engineering Research workshops and recent books on software business case and investment analysis.¹⁻³

Some EDSE contributions have explicitly addressed various aspects of software quality from value-based perspectives. These include Carnegie Mellon University's work on value-based security investment analysis⁴ and warranty models for software,⁵ the University of Virginia's application of real-options theory to the value of modularity⁶ and application of utility-theory and stochastic-control approaches to reliable delivery of computational services,⁷ and the University of Southern California's work on software cost and quality estimation models^{8,9} and on value-based software engineering processes, methods, dependency models, and tools.^{10,11} Springer's recent book, *Value-Based Software Engineering*,¹² contains a strong collection of additional contributions to VBSE theory, principles, and practices.

As an example VBSE application, value/risk-based testing emphasizes the testing objectives and plan derived from the original risk analysis, so that project decision-makers can prioritize test cases on the basis of prioritized requirements and operational scenarios. Project stakeholders can identify the risks that have been addressed (test cases passed) and the outstanding risks owing to either a lack of information (test cases that didn't run) or failed test cases.¹³

Relating test cases and results to business value and risk is particularly useful for projects with tight schedule and cost constraints.^{14,15} Value-based testing produces more business value per dollar invested generally than value-neutral testing techniques such as automated test generators, path testing, muta-

tion testing, or testing with unprioritized requirements. Future research challenges include finding value-based counterparts for these testing techniques.

References

1. D. Reifer, *Making the Software Business Case*, Addison Wesley, 2002.
2. M. Denne and J. Cleland-Huang, *Software by Numbers: Low-Risk, High-Return Development*, Prentice Hall, 2003.
3. S. Tockey, *Return on Software: Maximizing the Return on Your Software Investment*, Addison-Wesley, 2004.
4. S. Butler, "Security Attribute Evaluation Method: A Cost-Benefit Approach," *Proc. 24th Int'l. Conf. Software Eng. (ICSE 02)*, ACM Press, 2002, pp. 232-240.
5. P. Li et al., "The Potential for Synergy Between Certification and Insurance," Special ed. *ACM SIGSOFT, ICSR7*, 2002.
6. K. Sullivan et al., "Software Design as an Investment Activity: A Real Options Perspective," *Real Options and Business Strategy: Applications to Decision Making*, L. Trigeorgis, consulting ed., Risk Books, 1999.
7. Y. Cai and K. Sullivan, "Stochastic Optimal Switching," *4th Workshop Economics-Driven Software Eng. Research*, colocated with *Int'l Conf. Software Eng.*, 2002.
8. B. Boehm et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000.
9. B. Steece, S. Chulani, and B. Boehm, "Determining Software Quality Using COQUALMO," *Case Studies in Reliability and Maintenance*, W. Blischke and D. Murthy, eds., Wiley, 2002.
10. B. Boehm and L. Huang, "Value-Based Software Engineering: A Case Study," *Computer*, vol. 36, no. 3, 2003, pp. 33-41.
11. B. Boehm et al., "The ROI of Software Dependability: The iDAVE Model," *IEEE Software*, vol. 21, no. 3, 2004, pp. 54-61.
12. A. Aurum et al., eds., *Value-Based Software Engineering*, Springer, 2005.
13. R. Ramler, S. Biffl, and P. Gruenbacher, "Value-Based Management of Software Testing," *Value-Based Software Engineering*, A. Aurum et al., eds., Springer, 2005.
14. J. Bullock, "Calculating the Value of Testing," *Software Testing and Quality Engineering*, 2000, pp. 56-62.
15. P. Gerrard and N. Thompson, *Risk-Based E-Business Testing*, Artech House, 2002.

- project characteristics such as execution-time and storage constraints; and
- product characteristics such as complexity, reusability, and required reliability.

The parameters are calibrated to a diverse sample of 161 well-measured projects. The sample might not fully represent software projects in general, as many projects don't measure their performance.

In the regression analysis on the 161 projects, the projects reporting nominal required reliability (RELY) constitute the reference group. The relative effort for the projects with very high RELY rating (a product failure's impact was a potential loss of human life) is 1.26, and the relative effort for the ones with

very low RELY rating is 0.82 (a product failure's impact was a slight inconvenience). The ratio of the relative effort between the projects with very high RELY and very low RELY is 1.54. The regression analysis for the RELY variable produced a t-value of 2.602, well above the statistical significance level of 1.96 for this sample size and number of variables.¹

The added effort for a very high RELY project is the net result of two things: rework savings from early error elimination and extra effort in very thorough off-nominal, model-based, stress, and regression testing at the project's end. Near the project's end, project staffing is about at its average level, so the extra effort roughly translates into an extra 54 percent of calendar time in thorough testing before fielding the product.

Figure 1. Cocomo II software development cost-reliability-test time trade-off.

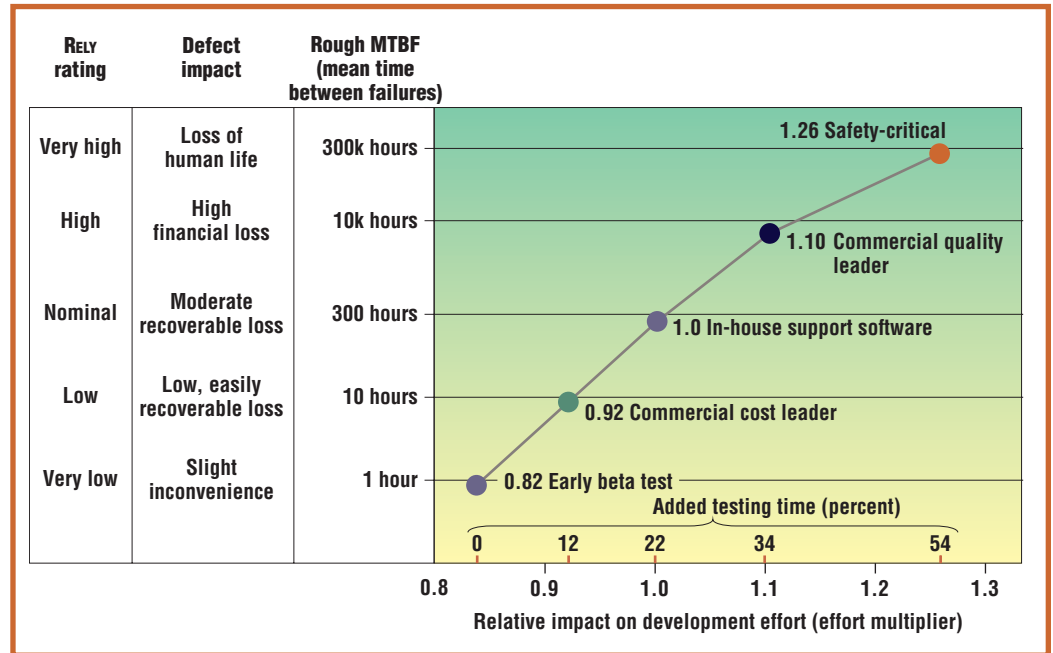


Figure 1 summarizes these results. On the basis of data from a subset of the 161 projects, we also added a rough mean-time-between-failures scale to the figure, corresponding to the relative impact of product failures. The scale goes from 1 hour MTBF for very low RELY to 300,000 hours MTBF for very high RELY.

Cost of reduced delivered-defect density: COQUALMO

As an extension of the COCOMO model, COQUALMO lets users first specify time-phased investment levels for improving software quality as measured by defect densities in requirements, design, and code, and then estimate the

resulting time-phased reliability levels. The current COQUALMO version estimates delivered defect density in terms of two models. A *defect-introduction model* estimates the rates at which the software requirements, design, and coding process phases introduce defects, and a *defect-removal model* estimates the defect-removal rates in these three phases.

The defect-introduction rates are a function of calibrated baseline rates that are modified by multipliers determined from the project's COCOMO II product, platform, people, and project attribute ratings. For example, a "very low" rating for "applications experience" will lead to a significant increase in the requirements defects introduced and to a smaller increase in the code defects introduced. The defect-removal model estimates the rates of defect removal as a function of the project's investment levels in peer reviews, automated analysis tools, and execution testing and tools. Its rating scales range from very low to extra high.

The calibrated baseline (nominal) defect-introduction rates for COQUALMO are 9 requirements defects/KLOC, 19 design defects/KLOC, and 33 code defects/KLOC. For simplicity and to avoid unwarranted precision, we've rounded these to 10, 20, and 30 for a total of 60 defects/KLOC introduced.² Starting from this baseline, figure 2 shows the COQUALMO estimation of reduced delivered-defect density as a function of the composite defect-removal

Figure 2. COQUALMO reduced delivered-defects estimates at nominal defect-introduction rates.

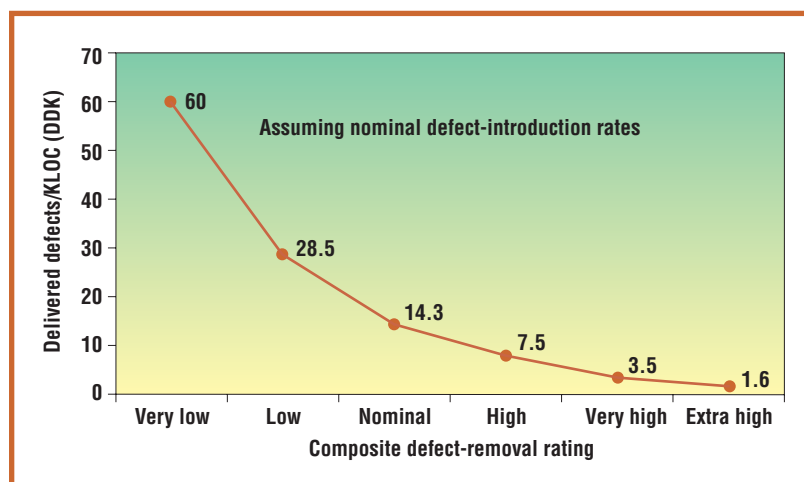


Table 1**Defect-removal investment rating scales**

Rating	Automated analysis	Peer reviews	Execution testing and tools
Very low	Simple compiler syntax checking	No peer review	No testing
Low	Basic compiler capabilities	Ad hoc informal walkthroughs	Ad hoc testing and debugging
Nominal	Compiler extension Basic requirements and design consistency	Well-defined sequence of preparation, review, and minimal follow-up	Basic test, test data management, problem tracking support Test criteria based on checklists
High	Intermediate-level module and inter-module Simple requirements and design	Formal review roles with well-trained participants, basic checklists, and follow-up	Well-defined test sequence tailored to organization Basic test-coverage tools and test support system Basic test process management
Very high	More elaborate requirements and design Basic distributed-processing and temporal analysis, model checking, and symbolic execution	Basic review checklists and root-cause analysis Formal follow-up using historical data on inspection rate, preparation rate, and fault density	More advanced test tools, test data preparation, basic test oracle support, distributed monitoring and analysis, and assertion checking Metrics-based test process management
Extra high	Formalized specification and verification Advanced distributed processing	Formal review roles and procedures Extensive review checklists and root-cause analysis Continuous review process improvement Statistical process control	Highly advanced tools for test oracles, distributed monitoring and analysis, and assertion checking Integration of automated analysis and test tools Model-based test process management

rating. The very low composite defect-removal rating leaves delivered-defect density at 60 delivered defects/KLOC, while an extra high rating can reduce the delivered-defect density to only 1.6 DDK.²

The composite defect-removal rating is an integration of the ratings for automated analysis tools, peer reviews, and execution testing and tools. It assumes nominal rates of defect introduction: a strong defect prevention program can reduce delivered-defect densities by another factor of 60 to 100. Developers can apply the RELY cost-estimating relationship to these investments. Table 1 correlates RELY's very low to very high rating levels to the horizontal rows of defect-reduction investments. For mixed investment levels in analysis, reviews, and testing, you can also determine the COQUALMO DDK estimates and an equivalent RELY rating.

Relationship between Cocomo II and COQUALMO

We've aligned COQUALMO rating scales for defect-removal investment levels via automated analysis, peer reviews, and execution testing and tools with the COCOMO II RELY rating levels shown in figure 1. The correspondence is based on a mapping between the activity analysis behind the COCOMO II RELY effort multiplier and the COQUALMO defect-removal activity ratings. Developers can thus

compare the investment levels for the low and high COCOMO II rating levels with the tools and activities assumed to be used at these levels in the COQUALMO rating scales.

This relationship between COCOMO II and COQUALMO also produces a way to relate investments in software reliability to resulting values of the delivered system's MTBF. For example, we've documented how to use this relation to determine how much availability is enough for various application classes.⁴

Value-estimating relationships

Finally, we need value-estimating relationships. A project's critical stakeholders must supply VERs by relating software quality levels or product delivery times to the resulting benefit flows and values earned. VERs assume that stakeholders have performed a baseline business case analysis for various value components (profit, customer satisfaction, on-time performance, and so on) as a function of software quality investment levels in peer review, automated analysis, and execution testing and tools. In the e-service domain, the major VERs involve losses in market share due to insufficient software quality or delayed product delivery.

VERs: Pareto distribution of requirement value

Most software testing research and tool building currently occurs in a *value-neutral setting*,

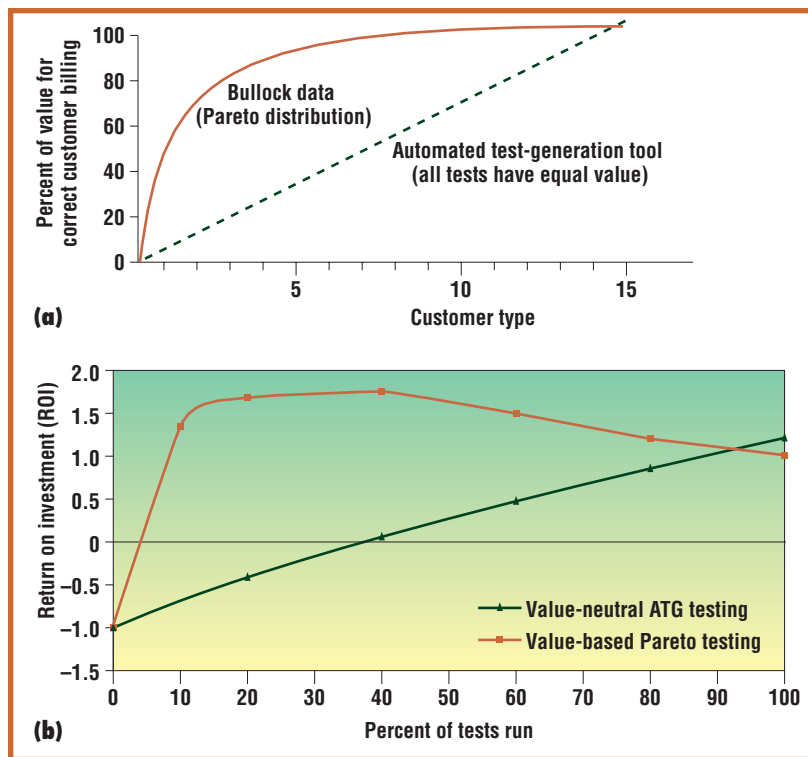


Figure 3. (a) Value-estimating relationships for value-neutral testing vs. value-based testing; (b) return on investment for value-neutral automated-test-generation testing vs. value-based Pareto testing.

Figure 4. Value loss (VL) vs. system delivery time (T_d): (a) marketplace competition, (b) fixed-schedule event support, and (c) offline data processing.

which considers every requirement, object, test case, and defect equally important. For example, the output from most automated-test generation tools is value neutral. With value-neutral testing, the earned mission value for invested testing effort will be linear (see the dotted line in figure 3a), because each requirement and test case is considered equally important.

However, in most operational situations,⁵⁻⁷ the value earned by each requirement will more likely follow a Pareto distribution (the solid curve in figure 3a). Accordingly, value-based testing focuses the testing effort on the roughly 20 percent of features that provide roughly 80 percent of the system value. For example, the empirical data in James Bullock's project ex-

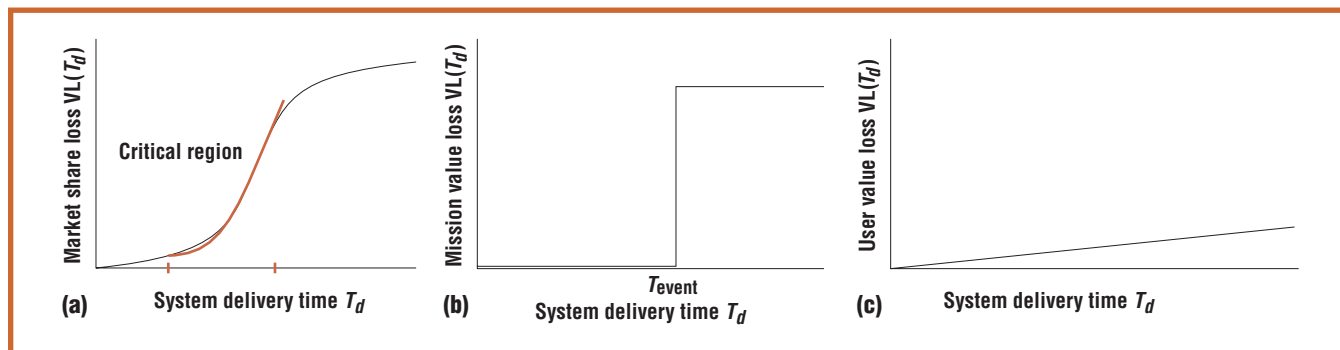
perience showed that testing each customer type improved billing revenues from 75 to 90 percent and that a single one of the 15 customer types accounted for 50 percent of all billing revenues.⁵ Thus, focusing initial testing on that one customer type provides an immediate boost in billing revenues per dollar invested in testing. As figure 3b shows, the resulting return on investment for value-based testing peaks at a considerably higher level than that for value-neutral testing.

There might nevertheless be good reasons, such as preserving good customer relationships, to continue testing after reaching the peak ROI. Furthermore, project decision-makers can often use the experience from high-value testing to make the selection of test cases or the use of test data generators for the lower-value requirements more cost-effective.

VERs: Value loss vs. system delivery time

The initial VERs for system delivery time show different types of stakeholder value-utility functions for relating the mission/market-value loss to the delivery time. These functions usually reflect the "cost of delay" in missed opportunities to meet time-critical commitments when a software system misses its delivery schedule. We describe three types of value-utility functions for system delivery time.

For marketplace competition, the value-utility function's shape is usually the classic S of economic production (see figure 4a). This shape generally characterizes delivered software capabilities such as e-services and wireless networking infrastructure. Early system delivery enables rapid capture of market share ahead of the competition. As the delivery time enters a critical region, significant competitors enter the marketplace and market share diminishes. The system value loss goes up rapidly until reaching



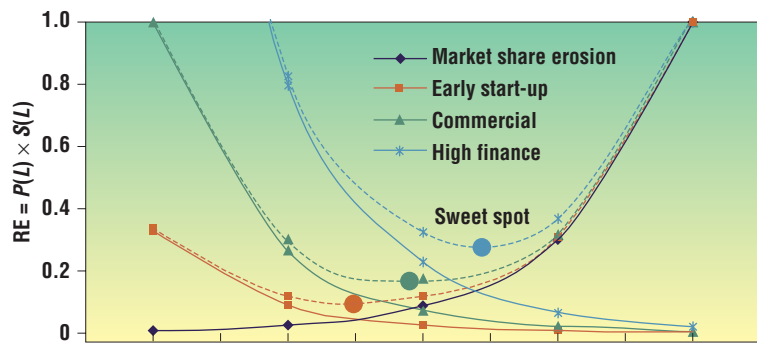


Figure 5. Combined risk exposures: Early-startup, commercial, and high-finance business cases.

	Very low	Low	Normal	High	Very high	RELY
COCOMO II	0	12	22	34	54	Added % test time
COQUALMO	1.0	0.475	0.24	0.125	0.06	$P_d(L)$
Early start-up	0.33	0.19	0.11	0.06	0.03	$S_d(L)$
Commercial	1.0	0.56	0.32	0.18	0.10	$S_q(L)$
High finance	3.0	1.68	0.96	0.54	0.30	$S_a(L)$
Market risk	0.008	0.27	0.09	0.30	1.0	RE_m

a diminishing-returns point, when very little market share is left to lose.

For fixed-schedule event support, the value-utility function for time delay assumes a step shape (figure 4b). Software systems that support, for example, the Olympic games, a trade show, or a Mars Rover launch window lose all value by missing the deadline.

For offline data processing systems, the user value loss versus system delivery time can be relatively flat. For example, scientific applications for processing astronomical information or historical archives will retain their value fairly uniformly as a function of delivery time (figure 4c).

Using Cocomo II, Coqualmo, and VERs

We can now integrate the Cocomo II and Coqualmo cost and quality estimation models with empirically based business VERs to perform combined risk analyses that can determine a project's approximate quantitatively optimal software quality investment level and strategy.

Combined risk analyses

In figure 5, we build up information from Cocomo II, Coqualmo, and the VERs to show what quality investment level is enough for various situations.

The Coqualmo estimate of delivered-defect density in figure 2 provides a basis for estimating the *probability of loss*, $P_q(L)$, whether fi-

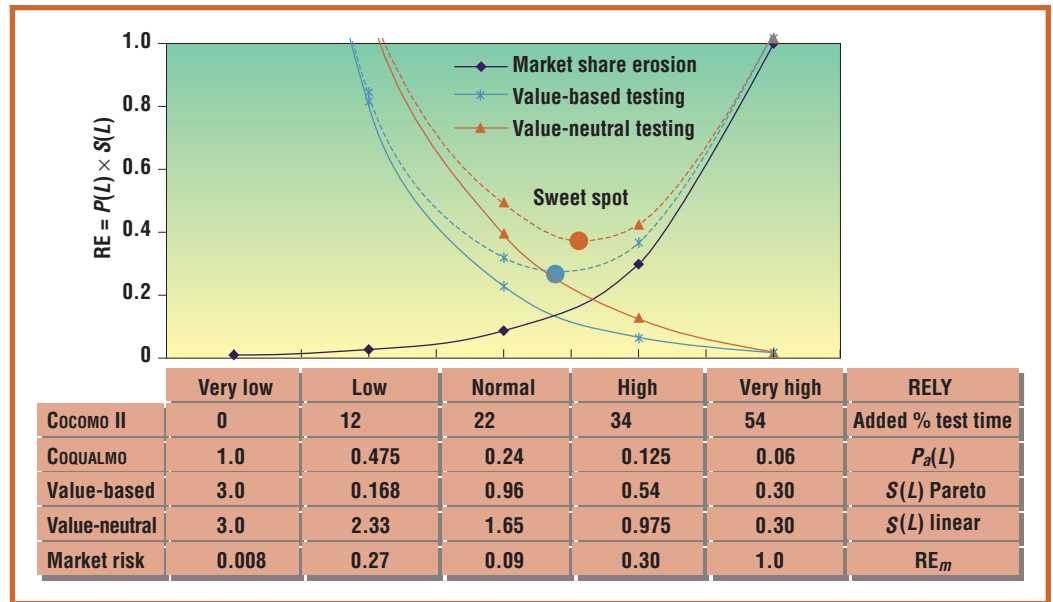
nancial or in terms of reputation and future prospects, that might result from unacceptably low quality: the fewer the defects, the lower the probability of loss. We can use the "very low" estimate of 60 defects/KLOC in figure 2 as the baseline for $P_q(L)$ and set its default value to 1. We can then compute the $P_q(L)$ for other RELY ratings, from low to very high, based on the corresponding delivered-defect density relative to the baseline, as shown in the second row of numbers at the bottom of figure 5.

We can obtain a baseline VER for the *size of loss*, $S_q(L)$, due to unacceptable quality for value-based testing⁵⁻⁷ from the Pareto distribution in figure 3a, using a negative Pareto distribution for value loss. Rows 3, 4, and 5 at the bottom of the figure 5 show this distribution and relative $S_q(L)$ for three representative business cases:

- early start-up (row 3), representing relatively defect-tolerant early adopters;
- normal commercial (row 4), representing the Bullock data;⁵ and
- high finance (row 5), representing very high-volume time-sensitive cash flows dependent on reliable operation of the software system.

For simplicity, we use a factor of 3 to distinguish the relative values of the three cases. Then we can compute the software quality investment risk exposure as $RE_q = P_q(L) \times S_q(L)$.

Figure 6. High-finance combined risk exposures: value-based testing vs. value-neutral testing.



With these values, we can calculate relative quality investment risk exposures as functions of added testing time for the three business case classes. Each of these stakeholder classes can then determine its own answer to the question, “How much software quality investment is enough?” by combining its software-quality-investment risk-exposure curve with its market-share-erosion risk-exposure curve RE_m . The latter comes from the critical region of the market share loss curve in figure 4a. For simplicity, figure 5 shows RE_m as a single curve (the line of diamonds) equal to 1.0 for a very high RELY rating and an added COCOMO-calibrated 54 percent delay in time to market. The value decreases by a factor of 0.3 for each successively lower RELY rating, as shown in the bottom line of numbers in figure 5.

Finally, we can find a *sweet spot* (the minimum) from the combined risk exposure of both unacceptable software quality and market-share erosion. Figure 5 shows the three combined risk-exposure curves in dashed lines and the corresponding oval sweet spots of software quality investment levels for the three business cases. For the high-finance case, the sweet spot occurs at the right-most side because the risk exposure of low system quality RE_q dominates. For the early-startup case, the sweet spot occurs at the left-most side because the risk exposure of high market-share erosion RE_q dominates. Such risk analyses can help project decision-makers determine the optimal stopping point in planning for “enough test-

ing” or, more generally, the optimal software quality investment level for their project based on their own business case.

The 1.0 baselining of the highest mainstream size of loss due to low software quality and the highest risk exposure due to market-share erosion means that the figure 5 model will adapt straightforwardly to other business situations. For example, software vendors in the high-finance market sector could replace the 1.0 baseline market-share risk exposure with their estimate of a US\$10 million loss in late delivery of a new feature in row 6 of figure 5 by multiplying the numbers in row 6 by \$10 million. Similarly, they could adjust the numbers in row 4 by replacing the 1.0 baseline business loss size in row 4 with an estimated \$30 million business loss of releasing a very low quality upgrade; this would generate a curve similar to the figure 5 star curve with a RELY investment sweet spot valued halfway between nominal and high.

Other analyses can similarly quantify software quality investment for other types of mission value loss reference points or alternative curves. Of course, determining absolute business values such as \$10 million and \$30 million might not be easy, particularly if the project hasn’t developed a business case. However, even relative values can help obtain useful decision insights.


Furthermore, we can use combined risk analyses to compare the results of value-based quality investment with value-neutral quality investment. Figure 6 presents the results for the

high-finance business case. As rows 3 and 4 at the bottom of the figure show, the decrease in $S_q(L)$ with testing time will be linear for value-neutral testing, while it will follow a negative Pareto distribution for value-based testing. The figure shows the combined risk exposure of value-based testing as the dashed line of triangles, while it shows the combined risk exposure of value-neutral testing as the dashed line of stars. The sweet spot of value-neutral testing moves up and to the right of that for value-based testing. In this example, RE_m for value-neutral testing is about 40 percent higher than it is for value-based testing. Of course, the project will need to invest in some form of early requirements prioritization, such as business case analysis, stakeholder win-win negotiation, total quality management, or agile methods story prioritization, but these generate other project advantages as well.

Value-Based Software Quality Model

To support these combined risk analyses, we've extended the iDAVE⁴ model into the Value-Based Software Quality Model. VBSQM provides the default values of $S_q(L)$ and of each RELY rating for three business cases we've described—that is, early start-up, normal commercial, and high finance. Users can also provide their own $S_q(L)$ values based on their own project business cases.

After users input the project size in KLOC and rate each COCOMO II cost driver, except RELY, according to their own project situation, VBSQM will generate the curve for combined risk exposure and help locate the sweet spot for the software quality investment level. In addition, because the iDAVE tool is spreadsheet-based, it's easy to modify it to handle other types of analyses using different VERs or to analyze the sensitivity of outcomes or the sweet spots to unavoidable uncertainties in the input parameters or value functions.

Even with only approximate information on relative values, the VBSQM we've described provides a framework to help reason about quality investment tradeoffs and decisions. Future research directions involve creating value-based counterparts for such value-neutral software quality technologies as test data generators, inspection checklists, defect closure metrics, and test plan aids. 

About the Authors



LiGuo Huang is a PhD candidate in the Computer Science Department at the University of Southern California and a research assistant in USC's Center for Software Engineering. Her research interests include value-based software engineering, software quality, high-dependability computing, software metrics and modeling, and software process modeling. She received her MS in computer science from USC. Contact her at the Center of Software Eng., Univ. of Southern California, 941 W. 37th Pl., SAL 330, Los Angeles, CA 90089-0781; liguohua@usc.edu.

Barry Boehm is the TRW professor of software engineering and director of the Center of Software Engineering at the University of Southern California. His research interests include value-based software engineering, software metrics and models, software process modeling, software requirements engineering, software architecture, and software engineering environments. Contact him at the Center of Software Eng., Univ. of Southern California, 941 W. 37th Pl., SAL 307, Los Angeles, CA 90089-0781; boehm@sunset.usc.edu.



Acknowledgments

This article is based on research supported by the US National Science Foundation. It was also supported by NASA's High-Dependability Computing Program contract to the University of Southern California. We express special thanks to *IEEE Software* associate editor in chief Stan Rifkin for his valuable improvement and editing suggestions in this work.

References

1. P. Crosby, *Quality Is Free*, McGraw Hill, 1979.
2. B. Boehm et al., *Software Cost Estimation with COCOMO II*, Prentice Hall, 2000; software and documentation available at http://sunset.usc.edu/research/COCOMOII/cocomo_main.html#downloads.
3. B. Steece, S. Chulani, and B. Boehm, "Determining Software Quality Using COQUALMO," *Case Studies in Reliability and Maintenance*, W. Blischke and D. Murthy, eds., Wiley, 2002; documentation available at http://sunset.usc.edu/research/coqualmo/coqualmo_main.html.
4. B. Boehm et al., "The ROI of Software Dependability: The iDAVE Model," *IEEE Software*, vol. 21, no. 3, 2004, pp. 54–61; initial spreadsheet tool available at <http://sunset.usc.edu/cse/pub/research/iDAVE/iDAVE.zip>.
5. J. Bullock, "Calculating the Value of Testing," *Software Testing and Quality Eng.*, vol. 2, no. 3, 2000, pp. 56–62.
6. P. Gerrard and N. Thompson, *Risk-Based E-Business Testing*, Artech House, 2002.
7. R. Ramler, S. Biffl, and P. Gruenbacher, "Value-Based Management of Software Testing," to be published in *Value-Based Software Eng.*, S. Biffl et al., eds., Springer, 2005.

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