Improving Software Quality as Customers Perceive It

Randy Hackbarth, Audris Mockus, John Palframan, and Ravi Sethi, Avaya Labs Research

// In this software quality improvement method, the Customer Quality Metric quantifies quality as customers perceive it. The Implementation Quality Index identifies projects that can improve their development processes. Prioritization tools and techniques focus limited resources on the riskiest files. //

WE FOCUS ON *customer quality*—quality as customers perceive it—in terms of how serious defects affect customers. As Watts Humphrey noted, "The cost and time spent in removing software defects currently consumes such a large proportion of our efforts that it overwhelms everything else."[1] Other quality aspects, such as whether a product does what customers expected, are outside this article's scope.

We've developed a customer quality method with three elements. First, the *Customer Quality Metric* (CQM)[2] is based on the fraction of systems reporting *customer-found defects* (CFDs). By "system," we mean an installation of a product at a customer site; we discuss CFDs in more detail later. Second, the *Implementation Quality Index* (IQI) evaluates error-removal practices during implementation; it predicts future postinstallation customer quality. Finally, prioritization techniques and tools focus limited resources on the riskiest files in the project's code repository.

Other metrics can be added—say, for testing practices—as long as they correlate with improved customer quality downstream. Furthermore, our method allows the addition of other prioritization or risk-management techniques.

This method has been adopted company-wide at Avaya, a global provider of business communication and collaboration systems. Avaya already had a strong commitment to quality when it faced quality issues with some of its products in 2011. With a strong executive focus and governance provided by Avaya's R&D Quality Council, the method has contributed to more than 30 percent year-over-year improvements in key metrics related to customer quality.

Customer-Found Defects

Not all defects are equal. Most defects are found and fixed during development or testing, before a product is delivered. Once the product is in use, customers have to observe an issue and care enough to report it for the issue to reach a support services organization. The issue must then survive various screening levels to be escalated to the development team. The team does its own screening before identifying the issue as a software defect. At Avaya, less than 1 percent of customer service requests materialize as CFDs (see Figure 1).

Field Quality: Customer Quality Metric

CQM represents the probability that a randomly chosen customer will be affected. We include a defect in a

 0740-7459/16/$33.00 © 2016 IEEE

particular system installation in our calculations even if the same defect was previously reported for a different installation. Having 10 systems reporting one CFD—even the same CFD—is worse than one system reporting 10 CFDs.

The fraction of affected systems better reflects the impact on customers than traditional product quality metrics[3] such as defect density and the number of CFDs. Defect density is a property of the code rather than the customer experience. Avaya's experience is that the number of CFDs measures the breadth of product deployment rather than the probability that a customer system will be affected.

Mature quality practices and a wealth of data make the Avaya Aura Communication Manager a good candidate for illustrating trends. On the basis of 272,000 system installations and upgrades of Communication Manager since 2002, releases with more installed systems have more CFDs. The number of CFDs increases linearly with the number of installed systems (see Figure 2a).

To facilitate quality comparisons across products and releases, we use two parameters. The *product maturity period* is the first m months after release (shaded in Figure 3). Quality improves as a product matures because early defects get fixed and later installations go more smoothly. At Avaya, m is usually 7 months.

The second parameter is the *postinstallation interval*, n (the solid lines in Figure 3). A longer interval captures more issues, thus more accurately indicating customer experience. However, it requires waiting longer after releases for the data. At Avaya, we estimate postrelease quality using one-, three-, and six-month intervals.

So, here's our formal definition of CQM: The n-month CQM is the

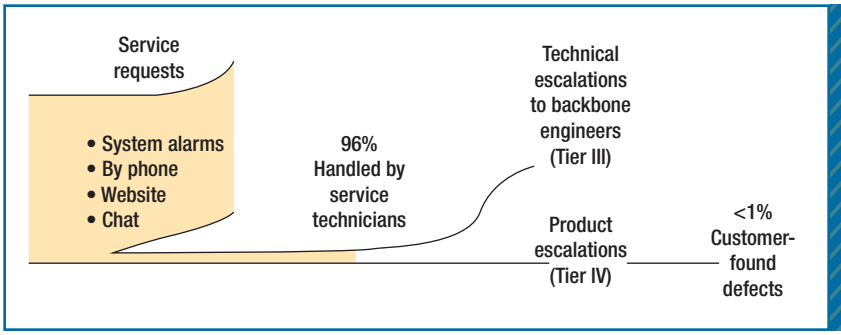


FIGURE 1. At Avaya, less than 1 percent of customer service requests are classified as customer-found defects (CFDs).

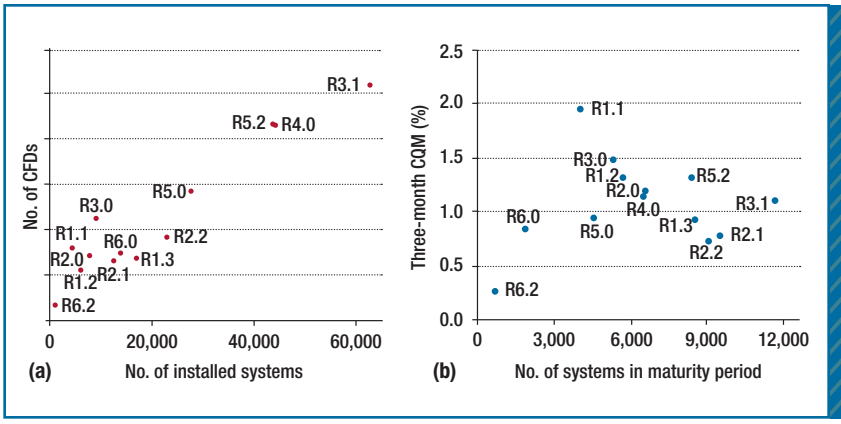


FIGURE 2. Data for releases of the Avaya Aura Communication Manager. (a) The number of CFDs compared to the number of installed systems. (b) The three-month Customer Quality Metric (CQM). Lower is better.

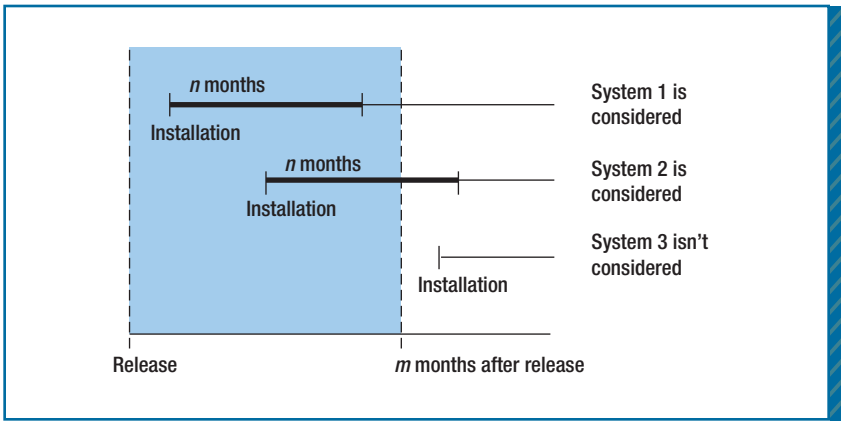


FIGURE 3. To be considered for the CQM, a system must be installed within the m-month product maturity period (shaded) and report a CFD within an n-month postinstallation interval. Avaya uses $m = 7$ and estimates postrelease quality at $n = 1$, 3, and 6 months.

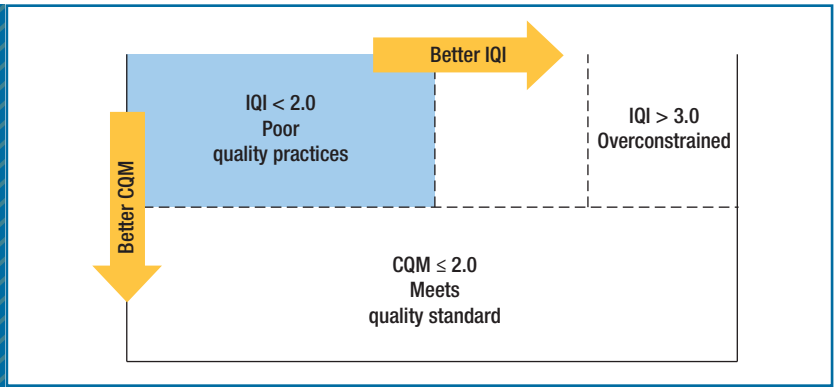


FIGURE 4. The implementation quality index (IQI) is scored on a scale of 0 to 4, where 4 is "done well," 2 is "done partially," and 0 is "done poorly or not at all."

fraction of systems installed within the first m months after release that have a trouble ticket leading to a CFD within their n-month postinstallation interval.

A lower CQM is better because it implies proportionately fewer defects. At Avaya, the current three-month CQM standard for new releases is 2 percent. The three-month CQM values for releases 1.1 through 6.2 of Communication Manager in Figure 2b were all below 2 percent. CQM values have been dropping at Avaya: 77 percent of tracked projects met this standard in 2013, up from 30 percent in 2011.

Error Removal: Implementation Quality Index

The IQI scoring mechanism helps guide development teams on where to invest proactively in error removal. Specifically, IQI measures the effectiveness with which the development project engages in four error-removal practices:

- static analysis, using industry-standard tools;
- code coverage (for example, developing unit white-box tests coincident with writing code and assuring adequate coverage by executing a code coverage tool);
- code reviews and inspections; and
- automated regression testing, primarily using black-box tests.

Each practice receives a score based on criteria specific to it. Scores range from 0 to 4, with 4 for "done well," 2 for "done partially," and 0 for "done poorly or not at all." IQI is the average of the scores for the individual practices.

The Avaya standard for IQI is 3.0 (see Figure 4). Although diminishing returns start to set in, individual teams perceive enough benefit that they're increasingly setting targets higher than 3.0. A score below 2.0 reflects poor practices.

The IQI practices are standard industry practices; their combination is known to be effective for error removal. From the benchmarking data provided by Capers Jones,[3] the combination of reviews, static analysis, and testing is 85 to 99 percent effective in error removal. The IQI practices relate to several CMMI level-3 process areas, such as Technical Solution (TS), Verification (VER), and Validation (VAL), as well as the level-2 process area, Measurement and Analysis (MA).

IQI Scoring

At Avaya, IQI scoring involves two stages. In the first stage, the projects themselves do the initial scoring. For that scoring, projects employ a standard template and detailed guidelines, specific to each practice, on what would be considered a top score (4), moderate score (2), or poor score (0). Table 1 summarizes the guidelines.

In the second stage, Avaya's R&D Quality Council often adjusts the initial score during a review, using probing questions such as

- How consistent is the scoring, compared to other projects?
- How effectively is the team engaging in the practices?
- Are they acting on the defects uncovered?

The resulting IQI score is accompanied by supporting comments that capture any concerns in plain English.

Correlation with Customer Quality

On the basis of our experience with more than 50 major projects, we've determined a positive correlation between improved development practices (higher IQI) and improved field quality (lower CQM) (see Figure 4). This empirical relationship justifies the time and effort spent in improving IQI.

Risk Mitigation

Simply providing information about the risk (high CQM) and suggested process improvements (through IQI scoring) wasn't enough: the projects needed help focusing their improvement efforts. So, we developed risk-prediction techniques and tools for prioritizing remediation actions.

TABLE 1

Guidelines for Implementation Quality Index (IQI) scoring.

Practice	Score* Green (4)	Yellow (2)	Red (0)
Static analysis			
Run regularly?	Part of the build process	Occasionally	Sparingly or not at all
Defects tracked?	Yes	No	No
High-impact defects corrected?	All	Most	Few
Code coverage (%)	≥ 75	≥ 50	< 20
Extent of code reviews	All new or changed code	Most code	Ad hoc or no reviews
Automated regression testing			
Percentage of tests	≥ 70	≥ 40	< 20
Investment	Ongoing	Much manual testing	Lacking

** A score of 4 means "done well," 2 means "done partially," and 0 means "done poorly or not at all."*

Risk Factors

The intuition that some parts of the source code are riskier than others isn't new:

- *Anecdotal evidence.* Robert Grady and Deborah Caswell recommended focusing on "the most complex modules."[4] Humphrey recalled a case at IBM in the 1980s in which "86 percent of the [1,600] modules had had no defects in three years. So 14 percent of the modules had all the defects, and 3 percent had half of them."[5]
- *Defect prediction.* Studies have shown that prior changes are a good predictor of postrelease defects.[6,7]

Practical applications of such predictions have lagged, however.[8] Risk prediction must be focused and accompanied with tool support. For example, "20 percent of the files" doesn't provide enough guidance for deploying limited resources.

On the basis of a regression analysis of historical defect data, the weighted sum of the following factors (over the previous three years) was a good predictor of risky files—files likely to have future CFDs:

- the number of past CFDs fixed in the file × 20,
- the number of file authors who have left × 10,
- the number of modification requests (MRs) × 0.1, and
- the number of unique versions × 0.01.

The weights of 20 for past CFDs and 0.01 for unique versions take into account that there can be orders of magnitude more versions than CFDs; see the examples in Figure 5.

For Avaya projects, we've found that the top 1 percent of files identified by this heuristic contribute to fixes to more than 60 percent of the CFDs. Audris Mockus and his colleagues described an earlier version of the risk predictor.[9]

Tool Support

The interactive risk-mitigation tool in Figure 5 supports problem discovery and resolution. It satisfies the diverse needs of the developers who fix the code and product managers who budget and schedule risk reduction.

The tool links risky-file analysis with code, developer, and organizational data. Code data includes individual files' source code, MRs, related files (files that were identical in the past to a candidate risky file), and other data from version control systems. Organizational data includes historical data from corporate directories. Historical data helps identify which authors have left the organization and when they left. From code data, we can infer each author's expertise with this and other files. MRs and CFDs provide helpful context to those who are evaluating what actions to take with each risky file. The risk-mitigation tool builds on the expertise browser.[10]

The tool accommodates the variety of defect amelioration approaches

CFDs by latest date (files by riskiest)	Modification requests	Authors	Related files
1) <PROJECT>/trunk/EPM/SMS/POManager/config/upgr/<FILE1.cpp>: 343 versions			
wi01079507 **2013-02-14** *CFD: ImportManager and import purge changes if there are a lot of completed import jobs …* wi00839993 **2010-12-09** *CFD: ftp import job stuck owing to invalid ip* 2 CFDs are 2% of the 78 MRs	78 MRs	19 authors 4 departed (21%)	6 files
2) <PROJECT>/trunk/src/mpp/media_svc/session_mgr/<FILE2.cpp>: 498 versions			
wi01162616 **2014-03-28** *Customer Feedback: Need document help file update and error message fix on Multitenancy* wi01051778 **2012-10-11** *CFD: Alarm management behavior on 6.0.1.0.0.0801* 2 CFDs are 1% of the 163 MRs	163 MRs 3 of "SDE" 2 of "Web Mgmt"	70 authors 30 departed (42%)	100 files of 180

FIGURE 5. A mock-up of a risk-mitigation tool that links analysis with code, developer, and organizational data. It satisfies the diverse needs of the developers who fix the code and product managers who budget and schedule risk reduction.

that Avaya was using. The nature of the risk and future development plans has led to policies such as these:

- Place high-risk areas into a control program in which changes are discouraged or, when necessary, require better inspections and testing.
- Assign owners for areas that are risky because of lost expertise. Give them sole responsibility for making most of the changes and overseeing others working in the area. This should build expertise and increase accountability.
- Consider refactoring or reengineering the riskiest areas that are expected to see much new development.

For one typical project, 16 files were candidates for control programs, and one file was identified for reengineering. Often, projects spread risk reduction work over several releases, starting with the easiest-to-implement steps, such as control programs and ownership or governance policies.

Between 2012 and 2014, the average CQM dropped from 2.9 percent to well below 1 percent, and the average IQI improved by 50 percent. Also, Avaya found that customer perceptions of product quality were a key contributor to customer satisfaction, measured by the net promoter score,[11] which increased by 60 percent.

People seeking to apply our method to their organization might proceed as follows. First, if needed, establish data collection about customer deployments, service alarms and requests, version control, code change information, and related organizational data.

Second, measure customer quality using a metric such as CQM. Analogous with testing, use a black-box customer quality metric rather than a white-box source code metric. As a reference point, the initial Avaya standard for CQM was 2 percent, as we mentioned before.

Third, once a customer quality metric is in place, use an in-process scoring mechanism such as IQI. This will help development teams improve their practices now, in anticipation of future improved customer quality, after the project is complete and systems are deployed at customer sites.

Finally, because risk is concentrated in a small fraction of the files, use a heuristic such as the one for risky files to focus improvement efforts.

Any quality improvement program needs governance and strong executive support. Our customer quality improvement method builds on an active research program to improve the state of software at Avaya, in partnership with the business groups. Measurable improvements in the state of software at Avaya accelerated after CQM and IQI became part of the corporate quality metrics.

Acknowledgments

David Weiss built up and led the Software Quality research program while he was with Avaya Labs Research. Evelyn Moritz created the Implementation Quality Index (IQI) and championed its use along with R&D Quality Council leaders Sarah Kiefhaber and Mike Jubenville. Jerry Glembocki, Saied Seghatoleslami, Dan Kovacs, and Lee Laskin were influential in establishing the corporate metric program

for the Customer Quality Metric (CQM) and IQI. Jon Bentley provided a careful and helpful review of this article. We also thank Avaya R&D leaders and team members who have embraced quality-focused software development, particularly IQI and CQM, as a way to track progress.

References

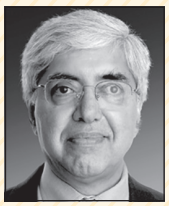
1. W.S. Humphrey, "Defective Software Works," Carnegie Mellon Univ. Software Eng. Inst., 1 Jan. 2004; www.sei.cmu.edu/library/abstracts/news-at-sei/wattsnew20041.cfm.
2. A. Mockus and D.M. Weiss, "Interval Quality: Relating Customer-Perceived Quality to Process Quality," *Proc. 2008 Int'l Conf. Software Eng.* (ICSE 08), 2008, pp. 723–732.
3. C. Jones, "Software Quality in 2013: A Survey of the State of the Art," 2013; http://namcookanalytics.com/software-quality-survey-state-art.
4. R.B. Grady and D.L. Caswell, *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, 1987.
5. W.S. Humphrey, *Oral History of Watts Humphrey*, 2009, Computer History Museum, ref. X5584.2010.
6. T.J. Yu, V.Y. Shen, and H.E. Dunsmore, "An Analysis of Several Software Defect Models," *Proc. IEEE Trans. Software Eng.*, vol. 14, no. 9, 1988, pp. 1261–1270.
7. T.J. Ostrand, E.J. Weyuker, and R.E. Bell, "Where the Bugs Are," *Proc. 2004 Int'l Symp. Software Testing and Analysis* (ISSTA 04), pp. 86–96.
8. E. Shihab et al., "High-Impact Defects: A Study of Breakage and Surprise Defects," *Proc. 2011 European Conf. Software Eng. and ACM SIGSOFT Symp. Foundations of Software Eng.* (ECSE/FSE 11), 2011, pp. 300–310.
9. A. Mockus, R. Hackbarth, and J.D. Palframan, "Risky Files: An Approach to Focus Quality Improvement Effort," *Proc. 2013 European Conf. Software Eng. and ACM SIGSOFT Symp. Foundations of Software Eng.* (ECSE/FSE 13), 2013, pp. 691–694.
10. A. Mockus and J. Herbsleb, "Expertise Browser: A Quantitative Approach to Identifying Expertise," *Proc. 2002 Int'l Conf. Software Eng.* (ICSE 02), 2002, pp. 503–512.
11. F. Reichheld and R. Markey, *The Ultimate Question 2.0: How Net Promoter Companies Thrive in a Customer-Driven World*, Harvard Business Rev. Press, 2011.

ABOUT THE AUTHORS

RANDY HACKBARTH coordinates a team at Avaya Labs Research dedicated to improving the state of the practice of software development at Avaya. Hackbarth received an MS in computer science and an MA in mathematics, both from the University of Wisconsin-Madison. He's a member of IEEE and ACM. Contact him at randyh@avaya.com.

AUDRIS MOCKUS is the Ericsson-Harlan D. Mills Chair Professor of Digital Archeology in the University of Tennessee's Department of Electrical Engineering and Computer Science. He's also a consulting research scientist at Avaya Labs Research. He studies developers' culture and behavior through the recovery, documentation, and analysis of digital remains. Mockus received a PhD in statistics from Carnegie Mellon University. Contact him at audris@avaya.com.

JOHN PALFRAMAN is a research scientist at Avaya Labs Research. He works with Randy Hackbarth on improving software practices at Avaya. Palframan received an M.Math from the University of Waterloo. He's a member of IEEE. Contact him at palframan@avaya.com.



RAVI SETHI is a professor in the University of Arizona's Department of Computer Science. He previously launched Avaya's research organization and was president of Avaya Labs Research. He's coauthor of the popular "dragon book" on compilers. Ravi received a PhD in computer science from Princeton University. He's an ACM Fellow and a member of IEEE. Contact him at rsethi@email.arizona.edu.