Customer Quality Improvement of Software Systems

Randy Hackbarth, Audris Mockus, John Palframan
Avaya Labs Research

Ravi Sethi
University of Arizona*

The software quality improvement method in this paper is based on a multi-year program to improve the quality of delivered systems at Avaya, a global provider of business communication and collaboration systems. The improvement method is data driven and has three elements: (a) a downstream metric that quantifies quality, as perceived by customers; (b) an upstream implementation quality index that measures the effectiveness of error removal practices during development; and (c) prioritization tools and techniques for focusing limited development resources. The downstream customer quality metric is based on serious defects that are reported by customers after systems are deployed. The upstream implementation quality index serves as a predictor of future customer quality; it has a positive correlation with the customer quality metric. The prioritization techniques are used to focus limited resources on the top 1% riskiest files in the code. Governance for the improvement method is provided by regular reviews with an R&D quality council.

Index Terms: Software quality method, customer perceived quality, data-driven software process improvement, software risk mitigation, case study

1 INTRODUCTION

Quality as perceived by customers – *customer quality* in short – is ultimately what matters for a software product. Let us refer to an installation of a product at a customer site as a *system*. We focus on serious customer reported defects (CFDs), rather than on other aspects of quality, such as whether a product does what customers expected. As Watts Humphrey notes, "the cost and time spent in removing software defects currently consumes such a large proportion of our efforts that it overwhelms everything else." [1]

* Ravi Sethi was with Avaya Labs Research when this work was done.

Avaya, a global provider of business communication and collaboration systems, has adopted a company-wide *customer quality method* that is data driven and has the following elements:

- A customer quality metric based on CFDs, after systems are deployed.
- A measure of error-removal practices during development, which is a predictor of future customer quality; it has a positive correlation with the customer quality metric.
- The metrics are accompanied by prioritization tools and techniques for focusing limited resources on the top 1% riskiest files in the project's code repository.
- Governance is provided by regular reviews with an R&D quality council.

Other metrics may be added – say, for testing practices – as long as they correlate with improved customer quality downstream. Further, the method allows other prioritization or risk-management techniques to be added.

At Avaya, the elements of the customer quality method were all in place when the company faced quality issues with some of its products in 2011. The quality issues prompted strong executive focus on the company-wide use of the customer quality method described in this paper, which contributed to 30+% year-over-year improvements in key customer quality metrics.

2 CUSTOMER FOUND DEFECTS

We reserve the term customer found defect (CFD) for the (small) fraction of customer service requests that have been thoroughly vetted by customer support and development personnel. Not all defects are equal. Most defects are found and fixed during development or unit 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 services organization. The issue must then survive various screening levels to be escalated to the development group. The development team does its own screening before identifying the issue as a software defect. At Avaya, less than 1% of customer service requests materialize as CFDs; see Figure 1.

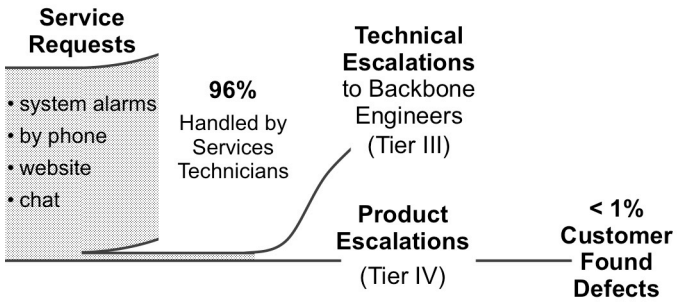

Figure 1. At Avaya, less than 1% of Customer Service Requests materialize as Customer Found Defects (CFDs).

3 FIELD QUALITY: CUSTOMER QUALITY METRIC (CQM)

For customer quality, we use a metric based on the fraction of affected systems because it represents the probability that a randomly chosen customer will encounter a defect. Lower probability is better. Thus, a few systems reporting many defects would be better than many customer sites affected by a few defects.

We have found that the fraction of affected systems better reflects customer expectations than traditional product quality metrics [2] like defect density and the number of customer found defects (CFDs). Defect density is a property of the code rather than customer experience. The number of CFDs measures the breadth of product deployment rather than the probability that a customer system will be affected. Figure 2(a) shows data for releases of the Avaya Aura® Communication Manager. Mature quality practices and a wealth of data make it a good candidate for illustrating trends. Based on 272,000 system installs and upgrades since 2002, the number of CFDs increases linearly with the number of installed systems. A similar pattern applies across other Avaya products.

In quantifying customer quality, we normalize for product maturity. Quality improves as a product matures, since early defects get fixed and later installs go more smoothly. In Figure 3, the maturity period (shaded), is the first m months after release; usually m is 7 months.

To facilitate comparisons across products and releases, we consider systems affected within the first n months after install (see Figure 3); in practice, the time intervals are one, three, and six months.

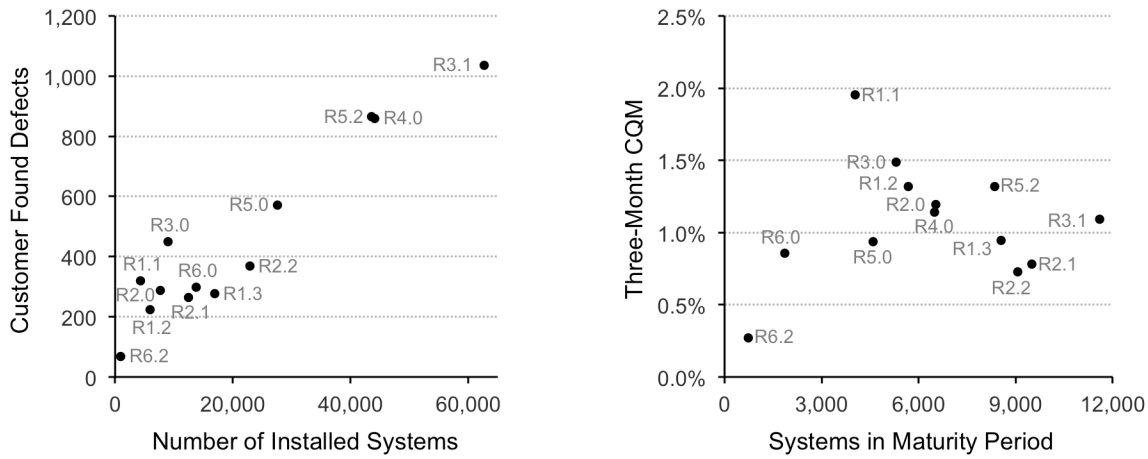

Figure 2. (a) Number of CFDs measures installs. (b) Three-Month CQM (lower is better).

Definition. The *n-month Customer Quality Metric (CQM)* [3] is the fraction of systems installed within the maturity period (the first *m* months after release) that have a trouble ticket within the *n*-month time interval after install, where the trouble ticket leads to a CFD.

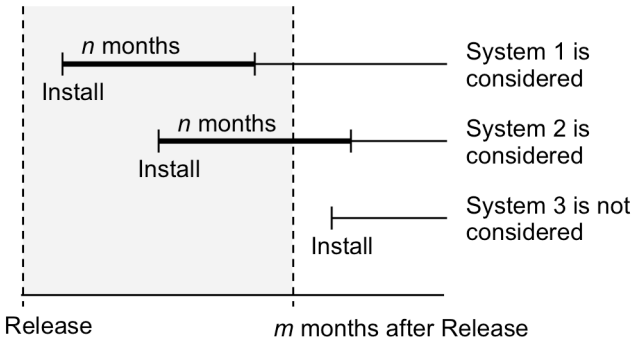

Figure 3. To be considered for the Customer Quality Metric, a system must be installed within the m-month maturity period after release (shaded). Trouble tickets from such a system are considered if they are within the n-month time interval after install.

Lower CQM is better, since lower implies proportionately fewer defects. Within Avaya, the current three-month CQM standard for new releases is 2%. The three-month CQM values for releases 1.1 through 6.2 of Communication Manager in Figure 2(b) are all below the 2% corporate standard. CQM values have been dropping in Avaya: 77% of tracked projects met this standard in 2013, up from 30% in 2011.

4 ERROR REMOVAL: IMPLEMENTATION QUALITY INDEX (IQI)

What development practices does a project need to improve today, in anticipation of improved customer quality in the future? The scoring mechanism for IQI (defined below) provides development teams with guidance on where to invest proactively in error-removal practices.

Definition. The *Implementation Quality Index (IQI)* for a development project is a measure of the effectiveness with which the project engages in four error-removal practices:

- Static analysis, using industry standard tools.
- Code coverage, e.g., developing unit "white-box" tests coincident with writing code and assuring adequate coverage via execution of a code coverage tool
- Code reviews and inspections.
- Automated regression testing, primarily "black-box" tests.

Each practice is assigned a score on a scale of 0-4; higher is better. Scoring is based on criteria specific to the practice, with 4 for "done well," 2 for "done partially," 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. An IQI score below 2.0 reflects poor practices. Diminishing returns set in above 3.0. Higher targets may constrain a project without commensurate benefit.

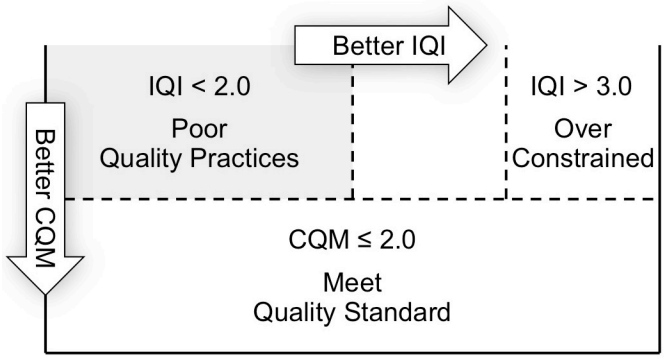

Figure 4. The Implementation Quality Index (IQI) is scored on a scale of 0-4.

The IQI practices themselves are standard industry practices. Their combination is known to be effective for error removal: from the benchmarking data provided by Capers Jones [2], the combination of reviews, static analysis, and testing is 85%-99% 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).

Scoring for IQI. At Avaya, scoring for IQI is done in two stages. The initial scoring is done by the projects themselves. For the initial scoring, projects are provided with a standard template and detailed guidelines, specific to each practice, on what would be considered a top score (4), a moderate score (2) or a poor score (0). See Table I for a summary of the scoring guidelines.

Table I. Guidelines for scoring practices for the Implementation Quality Index (IQI).

PRACTICE	GREEN (4)	YELLOW (2)	RED (0)
Static Analysis			
Run Regularly?	Part of build process	Occasionally	Sparingly or not at all
Defects Tracked?	Yes	No	No
Defects Corrected?	Significant defects	Most serious defects	Some serious or no defects
Code Coverage			
Percentage of code	≥ 75%	≥ 50%	< 20%
Code Reviews			
Extent of Reviews	All new/changed code	Most code	Ad-hoc or no reviews
Automated Regression Testing			
Percentage of Tests	≥ 70%	≥ 40%	< 20%
Investment	Ongoing	Much manual testing	Lacking

The project team's assessment is reviewed with and often adjusted by an R&D quality council, which probes to test the accuracy of the initial assessment.

- How effectively is the team engaged in the practices?
- How consistent is their scoring, compared to other projects?

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

Correlation with Customer Quality. Based on experience with over 50 major projects, we have observed 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.

5 A RISK MITGATION TOOL

We found that simply providing information about the risk (high CQM) and suggested process improvements (through IQI scoring) was not enough to help projects focus their improvement efforts. To help prioritization and remediation actions, we have developed a risk-mitigation tool that links code, developer, and organizational data.

Risk Factors. The intuition that some parts of the source code are riskier than others is not new.

- Anecdotal Evidence. Grady and Caswell recommend focus on "the most complex modules. [4]" At IBM in the 1980s, Humphrey recalls a case where "86 percent of the [1600] 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. A number of studies have shown that prior changes are a good predictor of post-release defects [6,7].

Practical applications of such predictions have lagged, however [8]. In addition to the need for tool support, risk prediction needs to be more focused: For example, "20% of the files" does not provide enough guidance for the application of limited resources.

We used regression analysis of historical defect data and identified the weighted sum of the following factors as a risk predictor of candidate risky files.

- Number of past CFDs (typically, trailing three years) fixed in the file × 20
- Number of file authors who have left × 10
- Number of modification requests (MRs) × 0.1
- Number of unique versions × 0.01

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

CFDs by LATEST DATE (FILES by RISKIEST)	MRs	AUTHORS	RELATED FILES
1) <PROJECT>/trunk/EPM/SMS/POManager/config/upgr/<FILE1.cpp>: 343 versions, 2680 LOC is 98% of max size			
wi01079507 **2013-02-14** *CFD:ImportManager and import purge changes if there are lots of completed import jobs …* wi00839993 **2010-12-09** *CFD:ftp import job stuck due 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.ccp>: 498 versions, 76 LOC is 100% of max size			
wi01162616 **2014-03-28** *Customer Feedback: Need document help file update and error message fix on Multi-Tenancy* 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 rendering of a risk-mitigation tool that links analysis with code, developer, and organizational data.

Our experience is that the top 1% of files identified based on this heuristic contribute to fixes to 60+% of the CFDs, for most Avaya projects. Mockus et al. describe an earlier version of the risk predictor [9].

Tool Support. The interactive risk-mitigation tool illustrated in Figure 5 supports both problem discovery and problem resolution. It satisfies the diverse needs of developers tasked with fixing the code and product managers tasked with budgeting and scheduling the risk-reduction effort.

At the developer level, the tool links risky file analysis with code, developer, and organizational data. Code data includes the source code of individual files, modification requests (MRs), related files (which are 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, so we can identify developers who are no longer with the organization. From the code data we can infer the expertise of each author 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].

We did not anticipate the variety of defect amelioration approaches that projects might use. Factors such as the nature of the risk and future product plans prompted simple rules like the following:

- Place high-risk areas into a "control" program where changes are discouraged or, when necessary, require better inspections and testing.
- Assign owners for areas that are risky because of loss of expertise. The owners will have sole responsibility for most of the changes and will oversee others working in the area. The intent is to build expertise and increase accountability.
- Consider refactoring or reengineering for the most risky areas that are expected to see much new development.

For one typical project, 16 files were candidates for control programs and 1 file was identified for reengineering. Often projects schedule risk reduction work over several releases, starting with the easiest-to-implement steps, such as control programs and ownership/governance policies.

6 DISCUSSION

The customer quality improvement method in this paper builds on an active research program to improve the state of software in Avaya. Some of our key learnings include the following.

- A governance program is critical to support and monitor deployment of a quality improvement initiative. An R&D quality council with strong executive support provides governance in Avaya and carefully tracks CQM and IQI scores as well as risky-file management.
- An IQI below 2.0 reflects poor development practices. We created a logistic regression model for the probability that a system would be affected by a customer-found defect (CQM ~ IQI), with IQI as a predictor, which shows that CQM (i.e., the probability of a customer observing a defect) increases 2.5 times with IQI dropping from 2.0 down to 1.0.
- The Avaya standard of 3.0 for IQI can be achieved by doing well with just three of the four error-removal practices, or by doing well enough on some combination of the practices; for example, by having static analysis and code coverage at 2.0 and code reviews and automated testing at 4.0. Keeping the IQI standard at 3.0 gives projects the flexibility to emphasize the practices that best meet their needs. Furthermore, we have not seen CQM benefits as a result of IQI scores greater than 3.0.

- The Avaya projects that benefit the most from improvements in IQI are the ones with starting CQM over 2%. The customer quality of the remaining projects, which have already achieved CQM≤2%, is essentially unaffected by further improvements in IQI. We believe that once a project has CQM at or below 2%, it has honed its development practices to the point where other external factors become dominant in determining customer quality.

We have received a number of practical suggestions for enhancing the Risky-File Management tool, which will be a focus of future work. For example, projects need to track, prioritize, and report on mitigation actions as part of their regular project activities. Some projects have requested that the results of such actions be reflected in the tool, along with qualitative input, such as an indication that there will be no future development in this area or a custom inspection checklist.

ACKNOWLEDGEMENTS

David Weiss built up and led the software technology research program while he was with Avaya Labs. Evelyn Moritz created the IQI metric and championed its use along with 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 CQM and IQI.

Jon Bentley provided a careful and helpful review of this paper.

We also thank Avaya R&D leaders and team members who have embraced quality-focused software development and the IQI and CQM metrics in particular as a means of tracking progress.

REFERENCES

1. W. S. Humphrey. 2004. Defective software works. Software Engineering Institute, Carnegie Mellon University. (January 2004). Retrieved January 16, 2014, http://www.sei.cmu.edu/library/abstracts/news-at-sei/wattsnew20041.cfm
2. C. Jones. 2013. Software quality in 2013: a survey of the state of the art. Retrieved January 17, 2014 from http:// http://namcookanalytics.com/software-quality-survey-state-art/
3. A. Mockus and D. M. Weiss. 2008. Interval quality: relating customer-perceived quality to process quality. In *International Conference on Software Engineering (ICSE '08)*. ACM Press, New York, NY, 723–732.
4. R. B. Grady and D. L. Caswell. 1987. *Software Metrics: Establishing a Company-Wide Program*. Prentice-Hall, Englewood Cliffs, NJ.
5. W. S. Humphrey. 2009. Oral history of Watts Humphrey. Interviewed by Grady Booch: June 17-22, 2009. Computer History Museum, Reference No. X5584.2010.

6. T. J. Yu, V. Y. Shen, and H. E. Dunsmore. 1988. An analysis of several soft ware defect models. *IEEE Transactions on Software Engineering* 14,9 (September 1988) 1261–1270.
7. T. J. Ostrand, E. J. Weyuker, and R. E. Bell. Where the bugs are. *International Symposium on Software Testing and Analysis (ISSTA '04)* ACM Press, New York, NY, 86–96.
8. E. Shihab, A. Mockus, Y. Kamei, B. Adams, and A. E. Hassan. 2011. High-impact defects: a study of breakage and surprise defects. In *European Conference on Software Engineering and ACM SIGSOFT Symposium on Foundations of Software Engineering (ECSE/FSE '11)*. ACM Press, New York, NY, 300–310.
9. A. Mockus, R. Hackbarth, and J. D. Palframan. 2013. Risky files: an approach to focus quality improvement effort. In *European Conference on Software Engineering and ACM SIGSOFT Symposium on Foundations of Software Engineering (ECSE/FSE '13)*. ACM Press, New York, NY, 691–694.
10. A. Mockus and J. Herbsleb. 2002. Expertise browser: a quantitative approach to identifying expertise. In International Conference on Software Engineering (ICSE '02). ACM Press, New York, NY, 503–512.

ABOUT THE AUTHORS

Randy Hackbarth:

Randy coordinates a team dedicated to improving the state of the practice of software development in Avaya . Before joining Avaya, Randy worked for 20 years at Bell Labs with a focus on establishing, coordinating and contributing to business unit - research partnership projects. He has an MS in Computer Science and an MA in Mathematics, both from the University of Wisconsin-Madison. Randy is a member of IEEE and ACM. Contact him at randyh@avaya.com.

Audris Mockus:

Audris Mockus received a B.S. in Applied Mathematics from Moscow Institute of Physics and Technology and a Ph.D. in Statistics from Carnegie Mellon University. He is affiliated with the University of Tennessee and Avaya Labs Research. Contact him at audris@avaya.com.

John Palframan:

John is a research scientist with Avaya Labs. He works with Randy on improving software practices in Avaya. Before joining Avaya, John was a technical manager in Bell Labs responsible for developing communications software and software development tools. He has organized annual software development conferences for the company since 1992. John is on the executive committee for the NJ Coast chapter of the IEEE. He has an M.Math and a B.Math (co-op) from the University of Waterloo. Contact him at palframan@avaya.com .

Ravi Sethi

Ravi Sethi launched the research organization in Avaya and was president of Avaya Labs, before joining the University of Arizona. He is an ACM Fellow and is a co-author of the popular "dragon book" on compilers. He has a B.Tech. from IIT Kanpur and a Ph.D. from Princeton. Contact him at rsethi@email.arizona.edu .