Beyond Defect Removal: Latent Defect Estimation With Capture-Recapture Method

Joe Schofield
Sandia National Laboratories

Defect removal and defect prevention techniques are no longer good enough to inspire confidence in software products. Techniques that help predict the number of remaining defects in software products can further boost customer confidence. Such techniques are easy to perform and have been used outside the realm of software engineering to produce reliable estimates for decades in the area of animal, bird, fish, and insect counts, and more recently for estimating the prevalence of the Severe Acute Respiratory Syndrome and cancer occurrences. This article describes the business case for removing defects and demonstrates how the usage of the Capture-Recapture Method (CRM) in defect removal activities can predict the number of estimated defects remaining in a product. This estimate can then be used to make quantified, data-driven decisions on how to proceed with a software product.

In December of 2005, Ford, Marriott, Sam's Club, and the Justice Department were all vilified in a nationally recognized information magazine for having customer data compromised through either theft or their inability to secure sensitive data [1]. Medical staff report that 770,000 medication mistakes occur each year in the U.S.; these errors are more than penmanship issues, transcription, data entry, and other preventable errors [2]. In 2004, interface issues between Hewlett-Packard's order entry system and SAP AG systems triggered $40 million in lost revenues [3]. Early in 2006, a property in Indiana valued at $121,900 had its value assessed for tax purposes at $400 million. The common thread to each of these incidents is software defects.

As recently as 2003, less than one-third of software organizations had a quality assurance group or processes [4]. Software developers like to use phrases like *level of rigor* and *quality commensurate with risk* to avoid or minimize the need for investing time in the quality of their products. Sound familiar? Tell the victims of the defects that it is *just a computer problem*, a *glitch*, an *issue*, a *foul-up*, a *snafu*, or a *bug*. Are they feeling better yet? What do you think is the level of confidence these victims have in the supplier? Will these consumers return and advocate the products and services they purchased?

Driving down the street we notice how credentialed the rest of our world has become. Attorneys, accountants, financial planners, physicians, surgeons, nurses, plumbers, electricians, engineers, and mechanics – they are all certified. But anyone with some level of educational or experiential hacking can write code. Credentials do not eliminate defects; verify this with a *certified* attorney. Credentials do however offer a measure of confidence to the consumer that the holder of the certification is trained and tested in the use of some body of knowledge, and often, subscribe to some code of ethics.

In lieu of certification credentials, another approach to raising the confidence of *software* consumers is to embrace defect removal and prediction techniques. The latent defect derivations that result from the prediction techniques are not rocket science. A peer recently taught fifth graders how to perform defect prediction; they became quite familiar with those techniques in merely a few hours.

> *"CRM affords a product development team the opportunity to employ statistical approaches to verify the goodness of a product as it is designed, developed, and deployed."*

Defects found during testing reveal as much about the adequacy of the process as they do the quality of the product. Is it not it an ominous sign when companies advertise that they are looking for more software testers? Clearly, quality (Q) without defect removal (Dr) is just faking (F) it (Q – Dr = F). But is the removal of *identifiable* defects adequate?

CRM affords a product development team the opportunity to employ statistical approaches to verify the *goodness* of a product as it is designed, developed, and deployed. Defect removal is woefully late and excessively costly during test (and even more so after release). CRM can be used by product teams to validate requirements and verify design criteria to reduce latent defects by estimating how many defects persist in their products. With this data, teams can make objective choices about proceeding or spending additional time to address unfound but predicted defects in their products. Eventually, practitioners benefit from the assurance of knowing that their products meet the expectations imposed upon them. Management benefits from the increased confidence that latent and hidden costs of post-delivery fixes are predictable, understood, and controlled. Ultimately, estimated latent defect data reduces the *risk* in risk management.

This article is not just another prognostication about a defect-induced apocalypse, nor is it another article to encourage more thorough testing to remove defects; after all, defect removal by testing is too similar to inspecting quality into a product as it rolls off the production line. This article is not about the effectiveness of inspections and peer reviews to remove defects close to their point of injection. So what, you might patiently ponder, *is* the purpose of this article? Not so fast.

Recently, a mid-level executive proudly shared that his team had just completed a one million line of code (1 MLOC) project with only 40 *issues* (notice the euphemism) reported. Ignoring the misunderstanding on his part regarding the significance of the size of the product [5], let us focus on the defects (issues) per MLOC. Forty deaths per million air miles or 40 injuries per million air passengers would not be acceptable to consumer safety groups. Forty deaths per year from providing wrong prescriptions is not healthy (the actual number is 7,000 per year) [6]. Forty cruise passengers returned to the wrong debarkation port would not float either. So why would 40 *issues* with a software delivery be hailed as laudable?

Does this statement reflect more about the expectations we have for software products or the state of maturity of software development in general?

While possibly more troubling or sensational, the above examples do provide perspective into the serious nature of defects of any kind. Incidentally, the 40-issue-defect-product above was a highly sensitive data collection system.

The lingering question in my mind was *how many defects have you and your customer not found, yet?* I knew he did not know, and I hardly wanted to ruin his otherwise sunny day.

So what is the purpose of this article? Simply stated, it is to encourage software engineers to use predictive techniques for determining the quality of products throughout their product development activities. The CRM is one such technique.

Brief Background

Our organization received a Capability Maturity Model for Software Level 3 certification in 2005. We rely on Personal Software Process℠ and Team Software Process℠ (TSP) as enablers of practice improvement. A colleague, Tom Cuyler, recently received his TSP certification. For the past year the organization has been re-engineering its software processes with a CMMI® Maturity Level 4 target. As part of our ongoing process improvement, Cuyler suggested we consider using the CRM which Watts Humphrey advocates in his TSP material [7]. Cuyler and I experimented with the CRM, he in his TSP work and I in our organization training.

We have collected defect data for the last five years. We know where our reported defects are injected, where they are detected, the defect type, its severity, the cost to repair, and the cost to discover (this last value is derived at a macro level). We derive and share defect leakage measures with project and management teams. We can estimate defects by function points in development and latent defects in delivered products. (Note: Latent defects can be estimated by defects reported by the customer after delivery using historical data from earlier projects. The defects not yet found by the customer, and perhaps never to be found remain unknown.)

So What's the Problem?

Defect riddled products continue to be released hindering the customer and casting a shadow of suspicion on the credibility of the supplier. Testing has not been effective in eliminating defects. Peer reviews and inspections have been effective in reducing, but not eliminating defects. Code testing tools cannot identify defects in the elicitation of requirements.

> *"Simply stated, [this article] is to encourage software engineers to use predictive techniques for determining the quality of products throughout their product development activities."*

To elaborate briefly, managers and project leaders have false confidence in product quality due to a paucity of the use of estimated latent defects in delivered products. In lieu of an approach like the CRM and statistical latent defect estimating (versus experiential or defect estimation based on *reported* defects), any claim about the quality of software is no more objective than that assertion from the aforementioned executive who deserved vigorous cross-examination.

And What's a Solution?

The CRM has been used for decades for sampling and estimating in disciplines unrelated to software engineering [8]. Even exploring the fine print and limitations of the technique, CRM is quite appropriate for peer reviews, for instance, (and even testing [if you must]). Caution: do not limit the use of CRM to peer reviews of code. Peer reviews and stakeholder reviews are useful mechanisms for verification and validation early in requirements capture, through design, as well as later during construction and testing. Here's a simple example of applying the CRM to a product that is being peer reviewed.

In Table 1, three product engineers identified a total of seven defects in a product; these are identified in the Defect Number column. In the next three columns, we associate which defects were found by which engineer in their individual preparation for the peer review. In Column A, the defects by the engineer who found the most unique defects are identified. In this case, Larry found the most unique defects, and Column A duplicates Larry's findings. In Column B, each defect that was found by all of the other participants is identified. In this case, the defects found by Curly and Moe are identified. In Column C, each defect that was found in both Column A and Column B are identified (e.g., the intersection of these two columns). The counts for Columns A, B, and C are totaled in this example, 5, 4, and 2, respectively.

The CRM indicates that the estimated number of probable defects in the product is:

(A * B) / C

in the example this value is:

(5 * 4) / 2 or 10

The CRM also indicates that the number of defects found by the participants is:

A + B - C

In the example this value is calculated as:

5 + 4 - 2 or 7

Finally, the CRM indicates that the estimated number of defects remaining is the difference between the probable number of defects (10) and the found defects (7) or 3. The *long hand* for this calculation is:

((A * B) / C) - (A + B - C)

For our example:

((5 * 4) / 2) - (5 + 4 - 2), i.e., 3

Table 1: *CRM Example*

Defect Number	Engineer Larry	Engineer Curly	Engineer Moe	"Column A"	"Column B"	"Column C"
1	✓			✓		
2	✓			✓		
3			✓		✓	
4	✓	✓		✓	✓	✓
5	✓			✓		
6	✓		✓	✓	✓	✓
7		✓			✓	
Totals	5	2	2	5	4	2

℠ Personal Software Process and Team Software Process are service marks of Carnegie Mellon University.

Therefore, in this example, the team has estimated that 70 percent of the defects in the product were identified as part of the peer review (and were/or will be removed), and that 30 percent of those defects remain.

Four important points are rendered here (The parenthetical references to CMMI are the most obvious mappings to the model and are not intended to be exhaustive.):

- First, the team has a quantified and objective process for determining the outcome of the peer review: repeat the review, accept the results of the review, or something else (CMMI Process Areas – Measurement and Analysis and Verification are supported with the CRM).
- Second, the team has an opportunity to establish defect removal thresholds – and manage to them. These thresholds could correspond to quality objectives for the organization and the project (CMMI Process Areas – Organizational Process Performance, Project Monitoring and Control, and Generic Practice 3.2 – Collect Improvement Information).
- Third, the estimated number of latent defects can be used to assess, analyze, and mitigate project risks (CMMI Process Area – Risk Management).
- Fourth, the outcome of any defect analysis can be used for improved training activities (CMMI Generic Practice 2.5 – Train People).

At a recent New Mexico Software Process Improvement Network (SPIN) meeting, Jerry Weinberg (the real Jerry Weinberg) was speaking about writing [9]. He referred to a manuscript which he had distributed to several associates. Weinberg indicated that he used the typos they reported to him to estimate the remaining typos in his document. I asked him if he used the CRM to do this, to which he responded (only slightly surprised by the question) *yes*. His writing project, in this case a book, was completed decades ago. Regrettably, the years erode the lessons and wisdom of the past.

Conclusion

CRM is widely used outside the software engineering world, and I suggest it is desperately needed inside the software engineering practices world. Easy, effective, and economical, we have found the CRM a valuable technique for quantifying confidence in products delivered. Stay tuned.◆

Acknowledgements

Thanks to Watts Humphrey for promoting this concept, to Tom Cuyler for introducing CRM in our community, to Jerry Weinberg for confirming its use outside of our initial research, and to Anna Nusbaum for her insightful review and edits of this article.

References

1. "December Data Exposures." Informationweek Feb. 2006: 19.
2. "Medication Systems." CIO June (2005): 28.
3. Levinson, Meredith. "The High Cost of Flawed Testing." CIO Nov. (2005): 66.
4. Surmacz, Joe. "Why Software Quality Stinks." CIO Dec. (2003).
5. "The Statistically Unreliable Nature of Lines of Code." CROSSTALK, Apr. 2005: 29-33 <www.stsc.hill.af.mil/crosstalk/2005/04/index.html>.
6. "When Did Six Sigma Stop Being a Statistical Measure?" CROSSTALK, Apr. 2006 <www.stsc.hill.af.mil/crosstalk/2006/04/index.html>.
7. Humphrey, Watts. Introduction to the Team Software Process. Addison-Wesley Professional, 2000.
8. LaPorte, R.E., D.J. McCarty., E.S. Tull, and N. Tajima. "Counting Birds, Bees, and NCDs." Lancet 1992.
9. Gerald Weinberg. <www.geraldmweinberg.com>.

About the Author

Joe Schofield is a distinguished member of the technical staff at Sandia National Laboratories. He is a Software Engineering Institute author and instructor for CMMI Introduction, is a trained Lean Six Sigma Black Belt, chairs the organization's software engineering process group, and leads the organization's movement from SW-CMM 3 to CMMI Maturity Level 4. Schofield chairs the Management Reporting Committee for International Function Point users Group, is active in the local SPIN, and has taught graduate level software engineering classes since 1990.

Sandia National Laboratories
MS 0661
Albuquerque, NM 87185
Phone: (505) 844-7977
Fax: (505) 844-2018
E-mail: jrschof@sandia.gov