Advancing Defect Containment to Quantitative Defect Management

Alison A. Frost and Michael J. Campo
Raytheon

The defect containment measure is traditionally used to provide insight into project success (or lack thereof) at capturing defects early in the project life cycle, i.e., the time when defect repair costs are at their minimum. Although the measure does provide insight into the effectiveness of early defect capture techniques (such as peer reviews), defect containment in its most common form (percentage of defects captured) is a lagging indicator as its ultimate value cannot be known until a project is complete. At that point, it is too late for a project to take corrective action. Using raw defect containment data and deriving Quantitative Defect Management (QDM) measures early in the development life cycle provides opportunities for a project to identify issues in defect capture before costs spiral out of control, schedule delays ensue, and another Death March begins [1].

Software quality issues have become a sad cliché in the software engineering industry. Versions 1.0 of commercial software products are notoriously defect-ridden. Furthermore, mission critical software has exhibited spectacular disasters, such as the loss of the Mars Climate Orbiter when English units were used in the coding of the ground software file used in trajectory models rather than the specified metrics units [2]. Ensuring software quality in mission critical systems is a primary cost driver in software development.

Several leading industry experts have analyzed defect injection rates during software development. Watts Humphrey found, " ... even experienced software engineers normally inject 100 or more defects per KSLOC [thousand lines of code] into their programs" [3]. Capers Jones gathered, "A series of studies found the defect density of software ranges from 49.5-94.6 errors per thousand lines of code" [4].

Compounding this situation, defects detected late in the development cycle cost many times more to repair than defects detected in the stage they were injected. For example, Watts Humphrey's research showed that the time it takes to fix a defect that escapes out-of-stage as shown in Table 1 [3].

Defect Containment Basics

Many companies employ a defect containment strategy in an attempt to reduce software costs and increase software quality. Programs and/or organizations may provide monthly resulting measures from this strategy as part of their team feedback or management reviews. Defect containment divides the engineering development cycle into separate stages and maps the stage in which a defect originated to the stage in which the defect was detected (see Table 2).

Defects may originate at any stage of the software development life cycle (although usually the greatest percentage of defects originates in the code and unit test stage). Defects detected in-stage are typically those defects detected during peer reviews or unit tests. Defects detected out-of-stage are those detected after the work product (e.g., design specification, or code) has been delivered to a downstream user (e.g., design released to development team or code released to software integration team). In-stage defects appear along the diagonal cells. (In Table 2, 2,421 defects originated and were detected during code and unit test.) Out-of-stage defects appear in the cells below the diagonal. (In Table 2, 1,525 defects originated in code and unit test, but were not detected until software integration.) These defect data provide insights to identify which processes cause the most defects and which processes allow defects to escape.

Defect containment is usually reported as percentages of defects captured in the stage in which they originated (see Table 3).

Using the data from Table 2, 48 percent of defects originating in design were detected (contained) in the design review process; 55 percent of defects originating in code were detected in the code review/unit test process; and the overall defect containment which equal the total number of defects caught in-stage/total defects for Table 2 was the following:

(1,515+1,555+2,421+37+1+10+0)/11,292 = 49 percent.

However, using defect containment to measure effectiveness as a percentage of in-stage capture is a lagging indicator. Until a project has gone through the later development stages, the ultimate number of defects injected is unknown. Furthermore, reporting superlative in-stage capture rates prior to qualification testing and system integration can be very misleading. The effectiveness of design and code peer reviews is unknown until it is too late for a project to take action. As such, traditional defect containment becomes a useful post-mortem tool, but does little to help a project when the project still has an opportunity to take corrective action.

Unfortunately, these two defect containment matrices (i.e. raw data count and percentage) are where the majority of engineers and managers conclude their defect data examination. However, by implementing a few derived measures from the defect containment base measures, one can employ proactive QDM.

Table 1: *Time to Fix Defect That Escapes Stage (in hours)*

Requirement	Design	Coding	Development Test	Acceptance Test	During Operation
1	3-6	10	15-40	30-70	40-1,000

QDM

QDM predicts the number of defects expected to be detected in each stage of software development, enabling proactive measures to be taken early in development. Why wait until system integration to discover that design and code peer reviews were ineffective? QDM allows a project to compare its defect detection rates against similar projects. These predictive and leading (as opposed to lagging) software measurements provide a mechanism to deter defect-driven cost and schedule overruns. This measure can be reported to a program and/or organization periodically (e.g. monthly) along with the defect con-

Stage Detected	Stage Originated							
	Requirements	Design	Code and Unit Test	SW Integration	SW Quality Test	System Integration and test	SW Maintenance	Total
Requirements	1,515							1,515
Design	1,181	1,555						2,736
Code and Unit Test	402	912	2,421					3,735
SW* Integration	200	420	1,525	37				2,182
SW Quality Test	191	223	370	7	1			792
System Integration and Test	89	114	114	5	0	10		332
SW Maintenance	0	0	0	0	0	0	0	0
Total	3,578	3,224	4,430	49	1	10	0	11,292

* SW = Software

Table 2: *Software Defect Containment Matrix*

tainment measures for team or management analysis and review.

Benefits of QDM are the following:

- Using predictive defect measures, a project knows in real time if it is meeting expected defect detection performance. For example, if a project is not finding the expected defects in design and code reviews, managers should investigate to determine if there is a reasonable cause or if corrective action is needed.
- Underperforming projects gain the ability to make corrective actions early rather than discovering problems at the end of the project.
- Overachieving projects provide the organization a chance to share best practices and lessons learned.
- Quantitatively understanding the capability of its peer review process offers an organization a chance to establish goals for defect capture and prevention, laying the groundwork for continuous improvement activities, and establishing Capability Maturity Model Integration (CMMI®) high maturity processes. (Please note: Although QDM may be a component of a CMMI high maturity process, by itself it may not qualify an organization to be rated CMMI Maturity Level 4 or 5.)

There are five key factors to take into account when applying QDM. To be effective, an organization must do the following:

1. Utilize *consistent definitions* for terms such as *defect*, *size unit* (e.g. source lines of code [SLOC]), and *life-cycle stages*.
2. Automate data collection and reporting to record and track defect data. Many change request tools exist that facilitate the recording and retrieval of in-stage and out-of-stage defect data, as well as automating the creation of derived measurement charts for projects. *Exploitation of automation allows projects to focus on data analysis rather than collection.*
3. Use *past data* to analyze current performance and predict future performance. Doing so allows one to create and maintain control limits based on performance capability. One can manage based on quantitative analysis.
4. Involve and train *all levels of personnel*. Besides improving data integrity, practitioners' perspectives and analyses are often found to be the most valuable. Ownership of organizational goals becomes shared by all levels of personnel.
5. Use QDM to improve project and organizational performance, *not to target individuals*. This is true of any measure.

QDM aligns with many industry initiatives. For example, QDM supports CMMI Level 4 and Level 5 as well as Six Sigma philosophies [5].

® CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

Table 3: *Software Defect Containment Percentage Matrix*

Stage Detected	Stage Originated						
	Requirements	Design	Code and Unit Test	SW Integration	SW Quality Test	System Integration and test	SW Maintenance
Requirements	42%						
Design	33%	48%					
Code and Unit Test	11%	28%	55%				
SW Integration	6%	13%	34%	76%			
SW Quality Test	5%	7%	8%	14%	100%		
System Integration and Test	2%	4%	3%	10%	0%	100%	
SW Maintenance	0%	0%	0%	0%	0%	0%	0%

In order to use defect containment in this predictive manner, organizations must do the following: 1) establish a baseline by using defect containment data from previously completed projects, 2) normalize the defect data found in each stage by size (e.g. SLOC), and 3) apply statistical techniques to set limits of expected defect detection performance.

Three QDM Measures

Three measures derived from the defect containment matrix that offer immediate proactive insight into defect data are the following: 1) Cumulative Defects Originated in Design Detected by Stage; 2) Cumulative Defects Originated in Code and Unit Test Detected by Stage; and 3) Defect Detection Distribution by Stage.

The following steps will cover required base measures, establishment of control limits/boundaries, and graphical representation of data. (More measures can be derived from defect containment to offer proactive insight, but this sample is a good start for a wide range of software engineers.)

In order to create these three QDM measures, the following base and derived measures are required:

- Number of defects by stage of origin (leveraged directly from the defect containment matrix [see Table 2]).
- Number of defects by stage of discovery (leveraged directly from the defect containment matrix [see Table 2]).
- A size count, such as SLOC or function points.
- Normalize the defect count by the size count (e.g. x defects per KSLOC).

Next, use comparison techniques on historical data to establish the range of expected defect detection, i.e., control limits. These data must come from independent observations of the same process (e.g. separate design reviews.) The data from each life-cycle stage is compared to data from its own stage. In cases where a defect in an earlier stage causes a defect in a later stage, the defect counts as a single defect in the stage it was originally introduced.

Control limits can be derived by calculating 3σ limits based on existing data. In this manner, defect data will fall between these (3σ) limits 99.7 percent of the time. Using 3-sigma limits avoids the need to make assumptions about the distribution of the underlying natural variation. As noted by Florac and Carleton in the following note:

> ... experience over many years of control charting has shown 3-sigma limits to be economical in the sense that they are adequately sensitive to unusual variations while leading to very few (costly) false alarms – regardless of the underlying distribution. [6]

Note: To calculate the control charts in these examples, the u-chart formulas were used.

For Cumulative Defects Originated in Design Detected by Stage (Figure 1) and for Cumulative Defects Originated in Code and Unit Test Detected by Stage (Figure 2), 3-sigma control limits are established using the following u-chart formulas:

$$UCL = \bar{u} + 3\sqrt{\frac{\bar{u}}{n_j}},$$

$$LCL = MAX\left[0, \bar{u} - 3\sqrt{\frac{\bar{u}}{n_j}}\right],$$

where ubar is the mean for each subgroup and n_j is the sample size. An example follows in Table 4. Note: In this example, KSLOCs were the size units of the design artifacts.

For Defect Detection Distribution by Stage (Figure 3), utilize a set of greater/less than boundaries. The number of defects detected in the software requirements stage should be less than the number found in the design stage. The number of defects detected should continue to increase through the code and unit test stage. After the code and unit test stage, the defects detected in each stage should decrease through the remaining stages with software maintenance stage detecting the least amount of defects.

Finally, plot the data.

For the Cumulative Defects Originated in Design Detected by Stage, create a chart where the x-axis is the life-cycle stage and the y-axis is the number of detected design-originated defects normalized by size unit. Plot the total normalized number of design defects found in-stage, followed by the total cumulative numbers of design defects detected in each subsequent life-cycle stage (code and unit test, software integration, software qualification test, system integration, and software maintenance). Pending analysis preference, data points may or may not be connected as a line on the chart; in the examples that follow, they are connected (see Figure 1).

For the Cumulative Defects Originated in Code and Unit Test Detected by Stage, create the charts similar to the process reviewed for Cumulative Defects Originated in Design Detected by Stage (see Figure 2).

For the Defect Detection Distribution by Stage, plot a chart where the x-axis is the software life-cycle stage and the y-axis is the normalized number of defects detected in each stage, regardless of the stage in which they were introduced (see Figure 3 for a display of the measure).

For Defect Detection Distribution by Stage, defect detection distribution ideally will mirror the defect injection distribution (thereby capturing defects as close as possible to when they were injected). It is known that the defect injection rate maps to the Rayleigh distribution curve as shown in Figure 4 [7]. (Statistically, the Rayleigh distribution is a Weibull Distribution with a value of two.) Therefore, it can be used to track the pattern of defect removal during the software life cycle.

Table 4: *Cumulative Defects Originated in Design Detected by Stage Control Limits*

Lifecycle Stage Where Design Defects Detected	Design Defects/Actual KSLOC	
	Minimum	Maximum
Design	3.7	9.9
Code and Unit Test	4.4	10.6
SW Integration	4.7	10.9
SW Quality Test	4.9	11.1
System Integration and Test	5.4	11.6
SW Maintenance	5.4	11.6

Analysis Results

Analysis of the QDM measures indicates the best course of action for the project and organization. Further, it is important to compare the QDM measures with other measures that the program or orga-

nization maintain in order to obtain a more complete understanding. Ultimately, the QDM measures provide indicators for further investigation. Opportunities to improve performance will vary among projects and organizations, as shown in the following examples:

- If a current project falls above the upper control limit, a course of action may be to perform causal analysis to understand the reason for the behavior. Possible actions include investigating means to reduce defects injected, adjusting control limits, and identifying best practices for defect detection to be considered for organizational deployment.
- If a current project falls below the lower control limit, a goal may be to get the current project to be as effective (e.g. during peer review) as the past projects; this would be demonstrated by moving the project within the control limits over time. In this case, the project may aggressively work to improve design and code peer reviews.
- Different opportunities exist if the project data falls within the control limits. Options include deploying defect prevention measures that drive the data toward the lower control limit of the charts illustrated in Figures 1 and 2. Alternatively, one may choose to gather a large enough sample to tighten the existing control limits and decrease projected variability.
- When looking at the Defect Detection Distribution by Stage measure, if the project has more defects detected in the design stage than the code stage (the defect detection efforts during code and unit testing may have not been effective), the project may not be ready to begin the software integration effort.

Establishing control limits on defect detection provides an organization the ability to predict the number of defects that will be inserted into project work products, based on work product size and the use of a standard organizational software development process. Predicting defects inserted within a statistically derived range may be used to determine readiness to move from one development stage to the next, and to predict future rework costs.

Further, utilizing organization data or industry standards on *hours to correct defects by stage*, return on investment can be calculated. Identifying peer review process or training issues can provide substantial savings for minimal investment.

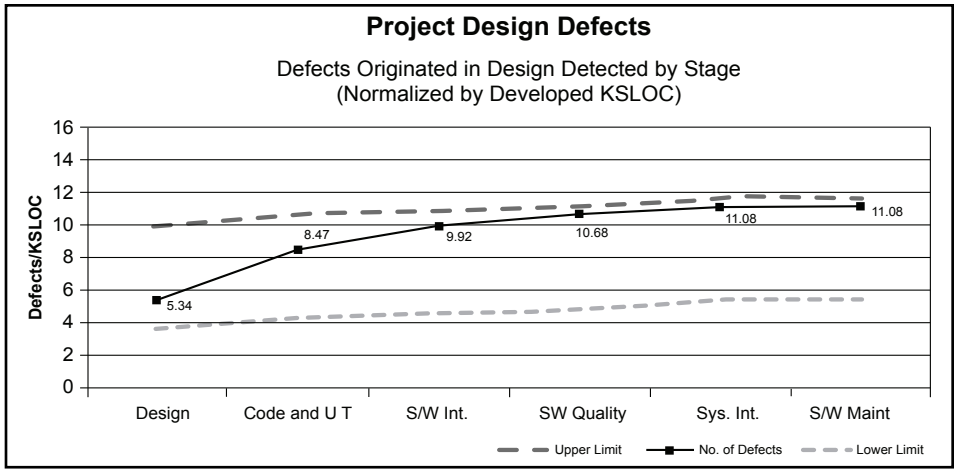

Figure 1: *Cumulative Defects Originated in Design Detected by Stage*

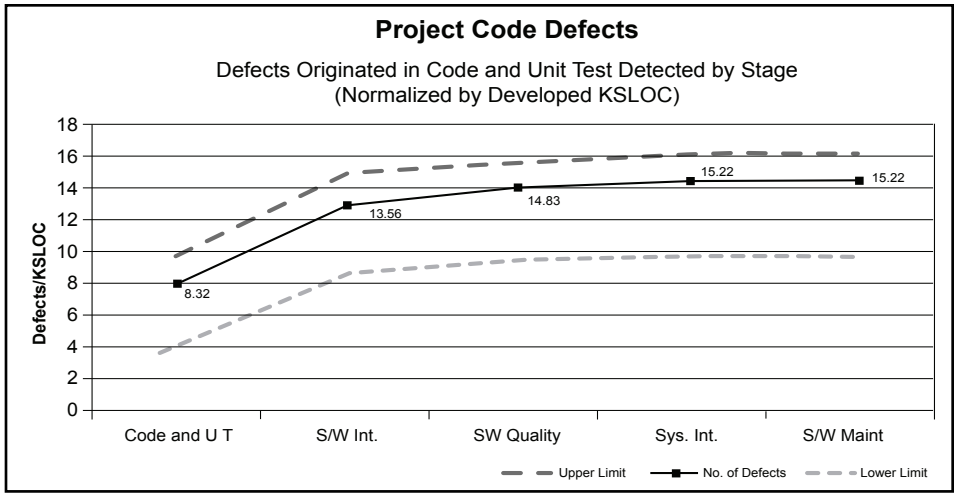

Figure 2: *Cumulative Defects Originated in Code and Unit Test Detected by Stage Chart*

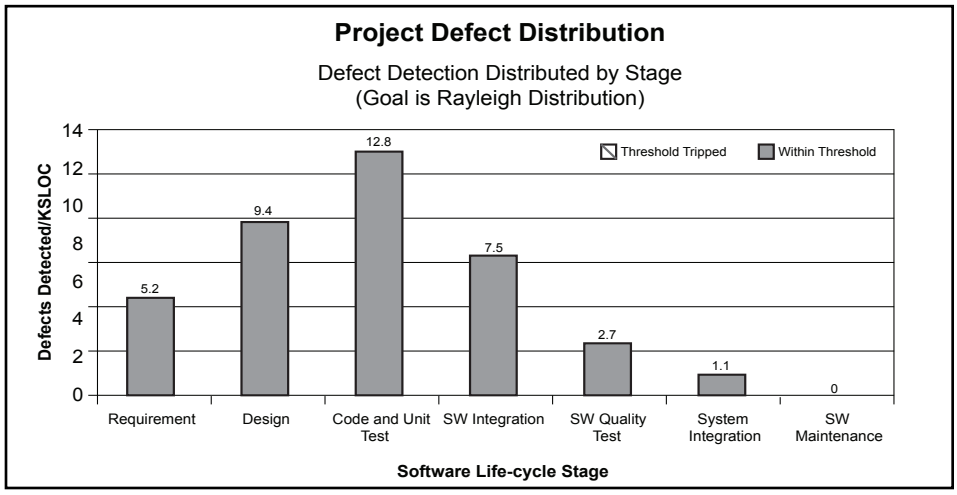

Figure 3: *Defect Detection Distribution by Stage*

Some examples of actual process improvements that resulted from the use of the QDM (implemented at Raytheon organizations) include the following:

- Design and code peer review standards were improved, with recommendations of:
  - o Design peer review preparation rate of less than 250 SLOC per hour per reviewer.
  - o Code peer review preparation rate

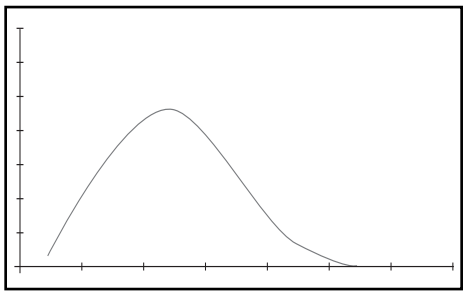

Figure 4: *Rayleigh Distribution*

of less than 200 SLOC per hour per reviewer.
  - Peer reviews meetings should not last longer than two hours [8].
- Peer reviews are postponed when participation is inadequate.
- Project meetings are held to provide feedback on QDM measures, address training, and investigate questions of data integrity.
- Software measurement tools were updated to improve automation of data collection and support analysis.

The improved peer review process, data entry and analysis, and measurement automation, were direct results of the QDM efforts.

Conclusion

QDM takes defect containment to a new level – from a reactive, lagging indicator to a proactive, predictive indicator of software quality. Samplings of derived defect measures with steps on how to create them were offered. QDM analyses provide an array of opportunities for process improvements that increase quality and reduce costs at both project and organizational levels.◆

References

1. Yourdon, E. Death March. 2nd ed. Prentice Hall, 2003.
2. Leveson, N. The Role of Software in Spacecraft Accidents. Massachusetts Institute of Technology, 2004.
3. Humphrey, W. "A Personal Commitment to Software Quality." Pittsburgh, PA: The Software Engineering Institute (SEI) <www.sei.cmu.edu>.
4. Jones, T.C. Programming Productivity. New York: McGraw-Hill, 1972.
5. SEI. Capability Maturity Model Integration. Version 1.2. Carnegie Mellon, SEI, 2006.
6. Florac, William A., and Anita D. Carleton. Measuring the Software Process. Addison Wesley, 1999.
7. Kan, Stephen H. Metrics and Models in Software Quality Engineering. Addison-Wesley Publishing Company, 1995.
8 Frost, A.A. "Design and Code Inspection Metrics." International Conference on Applications of Software Measurement, San Jose, CA, 1999.

About the Authors

Alison A. Frost is a national process engineer at Raytheon. While in Massachusetts, she was a leader in the achievement of CMM Level 4 for a 550-person Massachusetts/Alabama laboratory. Frost managed an engineering process group (EPG) metrics team comprised of statisticians, tools personnel, and systems/software engineers. In California, she was the DD(X) software measurement lead where she spearheaded the deployment of a measurement repository to more than 30 sites and companies. Currently, Frost is a Network Centric Systems Fullerton EPG member; her recent highlight was contributing the organization's CMMI software engineering/software/hardware Level 5 rating.

Raytheon
Fullerton, CA 92834
Phone: (978) 590-5905
Fax: (801) 340-7108
E-mail: frostfrost @earthlink.net

Michael J. Campo is a principal engineering fellow at Raytheon where he currently leads the Raytheon integrated defense systems (IDS) EPG and is a member of the Raytheon corporate CMMI expert team. Campo is a metrics leader at Raytheon. He is also an SEI-authorized CMMI Lead Appraiser and CMMI Instructor. He led Raytheon IDS to a CMMI Level 3 software engineering/software rating in 2003, and CMMI software engineering/software Level 4 and hardware Level 3 rating in 2005.

Raytheon
Tewksbury, MA 01876
Phone: (978) 858-5939
Fax: (978) 858-4505
E-mail: michael_j_campo @raytheon.com