Managing Software Quality With Defects[1]

David N. Card
Software Productivity Consortium

This article describes two common approaches to measuring and modeling software quality throughout the software life cycle so that it can be made visible to management. Both approaches involve developing a life-cycle defect profile, which serves as a "quality budget." This article also provides actual examples using each approach.

Many factors contribute to an increased practical interest in managing software quality. This means treating software quality as a key dimension of project performance, equal to cost (effort) and schedule. Corporate initiatives based on the Capability Maturity Model® (CMM®) [1], CMM Integration℠ (CMMI℠) [2], and Six Sigma [3] provide some examples of forces promoting an interest in quality as a management concern.

General management activities include planning, monitoring, and directing. In order to manage quality, it must be planned; accomplishment of the plan must be tracked, and appropriate corrective action must be taken as necessary. Nearly all projects establish budgets for effort and/or cost so that these dimensions can be managed. These budgets are plans for the expenditures of labor and/or dollars during the life of the project. Budgets typically identify planned total expenditures as well as expenditures during specific intervals such as life-cycle phases or months. Managing quality also requires establishing a budget for quality.

This article presents a simple approach to measuring and modeling software quality across the project life cycle so that it can be made visible to management. Next in the article are examples of applying this measuring and modeling approach in real industry settings. Both of the examples presented come from CMM Level 4 organizations. Whether or not the CMM or CMMI explicitly requires this type of analysis is beyond the intended scope of this article. More importantly, the approach has been shown to be practical and useful to project managers.

Software Quality and the Defect Profile

There are many views of software quality. The ISO/IEC 9126 [4] defines six:

- Functionality.
- Efficiency.
- Reliability.
- Usability.
- Maintainability.
- Portability.

Some of these quality factors are difficult to measure directly. Intuitively, the occurrence of defects is negatively related to functionality and reliability. Defects also interfere, to some degree, with other dimensions of quality. Both of the approaches discussed here involve developing a life-cycle defect profile. This defect profile serves as a *quality budget*. It describes planned quality levels at each phase of development just as a budget shows planned effort (or cost) levels. Actual defect levels can be measured and compared to the plan, just as actual effort (or cost) is compared to planned effort (or cost). Investigating departures from the plan leads to corrective actions that optimize project performance.

Software development consists of a series of processes, each of which has some ability to insert and detect defects. However, only the number of detected defects in each phase can be known with any accuracy prior to project completion. The number of defects inserted in each phase cannot be known until all defects have been found. Confidence in knowing that approximate number comes only after the system has been fielded. Consequently, this approach focuses on defects detected.

The techniques presented here depend on two key assumptions:

- Size is the easily quantifiable software attribute that is most closely associated with the number of defects. The basic test of the effectiveness of complexity models and other indicators of defect-proneness is to ask, "Does this model show a significantly higher correlation with defects than just size (e.g., lines of code) alone?" [5].
- Defect insertion and detection rates tend to remain relatively constant as long as the project's software processes remain stable. While the rates are not exactly constant, they perform within a recognized range.

The first assumption appears to be inherent to the nature of software. CMM Level 4 organizations actively work to make the second assumption come true. That is, they are acting to bring their processes under control.

An Empirical Model

The simplest approach to generating a defect profile for intended projects within the organization is to collect actual data about the insertion and detection rates in each life-cycle phase. This can be accomplished in the following four steps:

- First, historical data are collected. Table 1 shows a simple spreadsheet used to tabulate defect discovery and detection data using example data. In addition, the size of the project from which the defect data is collected must be known. The size measure must be applied consistently, but this approach does not depend on using any specific measure. Lines of code, function points, number of classes, etc., may be used as appropriate. (The data in Table 1 are simulated, not real.)
- Second, an initial profile of the number of defects found in each phase is gener-

Table 1: *Example of Empirical Defect Profile (Simulated Data)*

	Phase Inserted					
Phase Detected	Analysis	Design	Code	Developer Test	System Test	**Total**
Analysis	0					0
Design	50	200				250
Code	50	100	300			450
Developer Test	25	50	150	0		225
System Test	18	38	113	0	0	169
Operation	7	12	37	0	0	56
Total	150	400	600	0	0	1,150

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office.
℠ Capability Maturity Model Integration and CMMI are service marks of Carnegie Mellon University.

ated as shown in Figure 1. The bars in that figure represent the totals in the last column of Table 1.

- Third, this initial profile is scaled to account for differences between the size of the project(s) from which the profile was developed and the size of the project to which it is applied. This is accomplished by multiplying by the ratio of the project sizes. For example, if the defect profile in Figure 1 were to be used to develop a defect profile for a project twice the size of the project providing the data that went into Figure 1, then the bars of the profile representing the new project would be twice the size of those in Figure 1.
- Fourth, the scaled defect profile is adjusted further to reflect the planned performance of the project. For example, if the project plan called for the automatic generation of code from design instead of hand coding as previously done, then the number of defects inserted in the implementation phase would be adjusted downward to reflect this change in the coding process. Also, changes in the project's process may be induced in order to reach a specified target in terms of delivered quality if previous performance did not yield the required level of quality. The target might be specified as a result of a customer requirement or an organizational goal.

Actual defect counts can then be compared with this final plan (defect profile) as the project progresses. Suggestions for this activity are provided in a later section of this article. Note that the defect profile does not address defect status (i.e., *open* vs. *closed* problems/defects). All detected defects, regardless of whether or not they ever get resolved, are included in the defect counts.

Figure 2 shows an example of a defect profile developed empirically [6] for an actual military project. This figure shows the predicted number of defects to be injected and detected in each phase, based on previous projects. However, only actual counts are shown for the number of defects detected, because the actual number injected cannot be determined with any confidence until after software delivery.

The project in Figure 2 was about two-thirds of the way through software integration at the time data were reported. Two-thirds of the predicted number of defects had been found in software integration. The project's quality performance was tracking the plan. This illustrates that the real value of the defect profile lies in its ability to make quality visible during development, not as a post-mortem analysis technique.

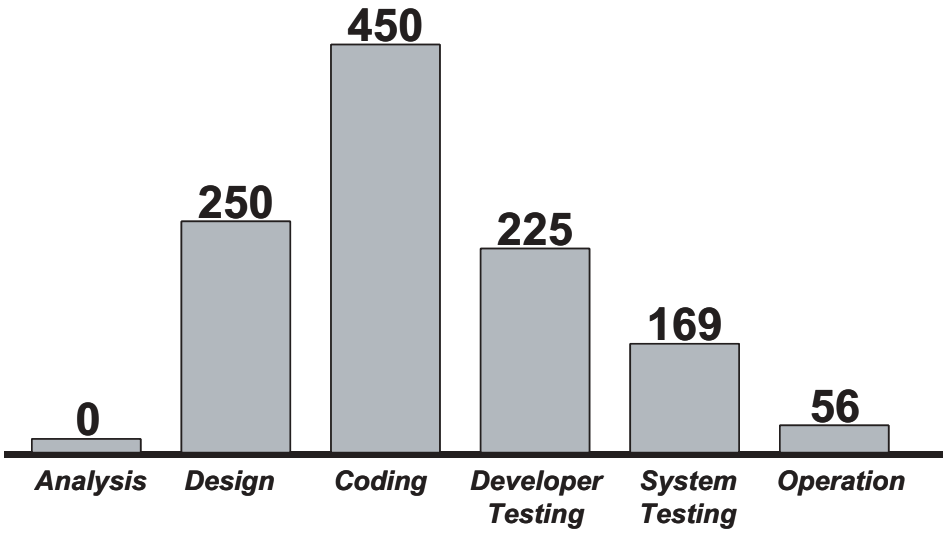


Figure 1: *Example of Defect Profile With Data From Table 1*

The project in Figure 2 actually was completed after this graph was prepared. The planned and actual defect levels never differed by more than 10 percent. The project team handed their product over to the customer with a high degree of confidence that it met the targeted level of quality.

An Analytical Model

Defect profiles may also be generated analytically. Many early studies of defect occurrence suggest that they followed a Rayleigh dispersion curve, roughly proportional to project staffing. The underlying assumption is that the more effort expended, the more mistakes that are made and found.

Gaffney [7] developed one such model:

$$Vt = E\,(1 - \exp(-B(t^{**}2)))$$

Where:

Vt = Number of defects discovered by time *t*.

E = Total number of defects inserted.

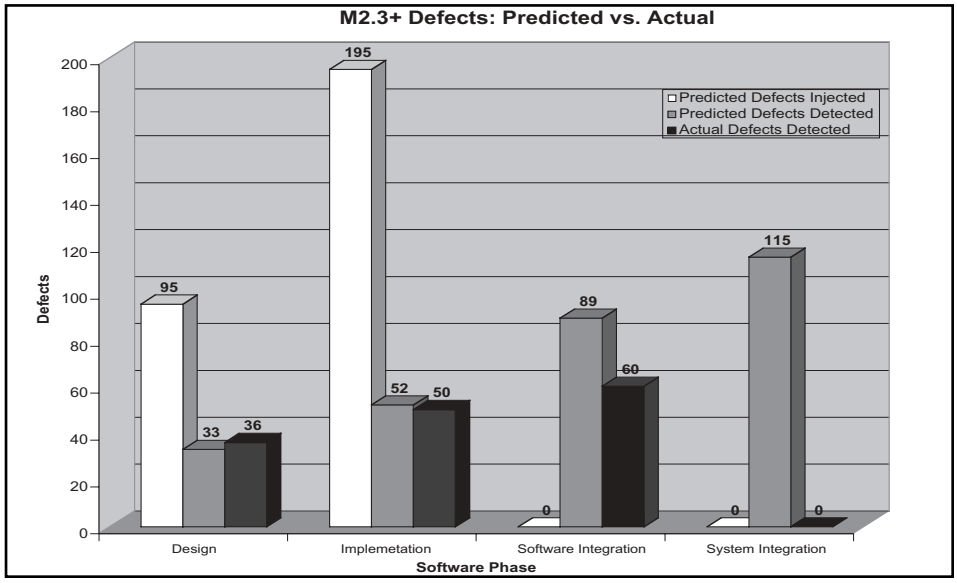
B = Location parameter for peak.

The time periods *t* can be assumed to be equal to life-cycle phase transition boundaries in order to apply the model to project phases rather than elapsed time. The location parameter *B* fixes the time of the maximum (or peak) distribution. For example, *B=1* means that the peak occurs at *t*=1.

The analytical approach involves applying regression analysis to actual phase-by-phase defect data to determine the values of *B* and *E* that produce a curve most like the input data. Many Software Productivity Consortium member companies use our proprietary software, SWEEP [8] (based on the Gaffney model), to perform this analysis, but it can easily be implemented in Microsoft Excel.

The effectiveness of the analytical approach depends on the satisfaction of additional assumptions, including the following:

Figure 2: *Example of Empirical Defect Profile (Actual Project Data)*



- Unimodal staffing profile.
- Life-cycle phases of similar (not exactly equal) duration (not effort).
- Reasonably complete defect reporting.
- Using only observable/operational defects.

To the extent that these assumptions are satisfied, this model gives better results. Analytical models such as this are useful when the organization lacks complete life-cycle defect data or desires to smooth existing data to provide an initial solution for new projects without prior historical data. The defect profile obtained from the actual data can be easily adjusted to fit projects with different numbers of life-cycle phases and processes by selecting appropriate values of *E* and *B*.

Figure 3 provides an example of a defect profile for another actual military project generated by SWEEP. The light bars in Figure 3 represent the expected number of defects for each phase, based on the model. For this specific project, the actual number of defects discovered is substantially lower than planned during design. Consequently, additional emphasis was placed on performing rigorous inspections during code, with the result that more defects than anticipated were captured during code, putting the project back on track to deliver a quality project as shown at post release.

A detailed discussion and analysis of applying the Gaffney model to a military project using SWEEP can be found in [9].

Interpreting Differences

During project execution, planned defect levels are compared to actual defect levels. Typically, this occurs at major phase transitions (milestones). However, if a phase extends beyond six months, then consider inserting additional checkpoints during the phase (as in the example in Figure 1 where analyses were conducted at the completion of each one-third of integration testing). Since real performance never exactly matches the plan, the differences must be investigated. This involves three steps:

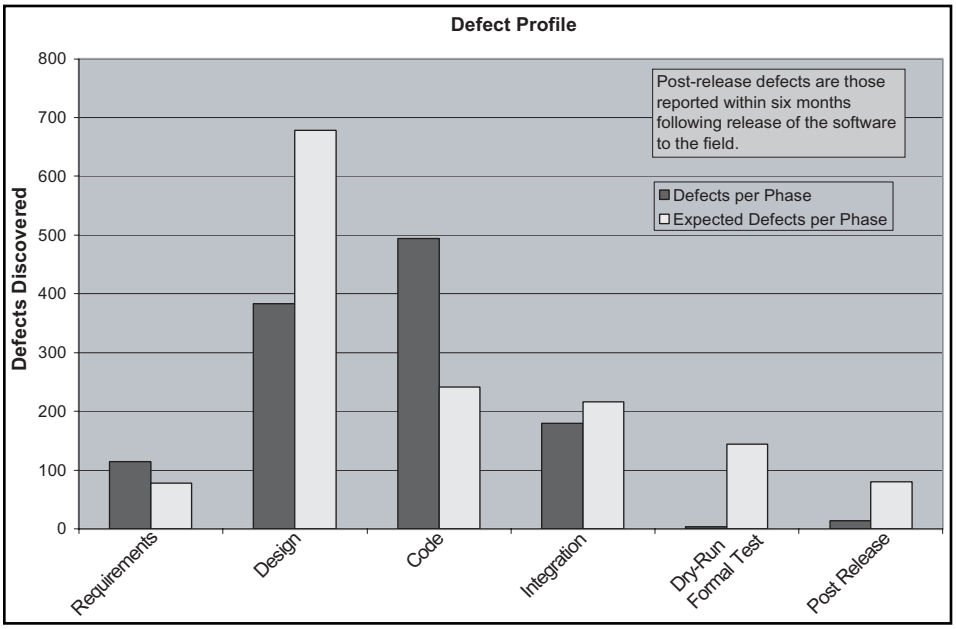
- Determine if the differences are significant and/or substantive. This might be accomplished by seeking visually large differences, establishing thresholds based on experience, or applying statistical tests such as the Chi-Square [10].
- Determine the underlying cause of the difference. This may require an examination of other types and sources of data such as process audit results as well as effort and schedule data. Many techniques have been developed for causal analysis (e.g., [11]), but they fall beyond the scope of this article.
- Take appropriate action. This includes corrective actions to address problems identified in the preceding step, as well as updates to the defect profile to reflect anticipated future performance.

Differences between planned and actual defect levels do not always represent quality problems. Potential explanations of departures from the plan include the following:

- Bad initial plan (assumptions not satisfied, or incomplete or inappropriate data).
- Wrong software size (more or less than the initial estimate).
- Change in process performance (better or worse than planned).
- Greater or lesser software complexity than initially assumed.
- Inspection and/or test coverage not as complete as planned.

Analyzing departures from the defect profile early in the life cycle provides feedback for our understanding of the size and complexity of the software engineering task while there is still time to react.

Figure 3: *Example of Analytical Defect Profile (Actual Project Data)*



Summary

Relatively simple models of software quality based on defect profiles are becoming increasingly popular in the software industry as organizations mature. These models establish a *quality budget* that helps to make trade offs among cost, schedule, and quality visible and reasoned, rather than choices made by default. Defect profiles present quality performance to the project manager in a form that he or she understands. Thus, the consequences of a decision such as "reducing inspection and testing effort to accelerate progress" can be predicted. Unintended departures from planned quality activities can be detected and addressed.

Moreover, the ability to model quality across the project life cycle is a necessary prerequisite to implementing design for Six Sigma techniques [3] in software development. Achieving Six Sigma requires measuring and managing quality at each software production step, not just during the final testing stages prior to delivery.

Defect models can become very rich. The concept of orthogonal defect classification [12], for example, involves developing separate profiles for each of many different defect types. These defect classifications facilitate the causal analysis process when potential problems are identified.

This article discussed two very simple approaches to building and using defect profiles. These techniques make quality visible so that it can be managed.◆

References

1. Paulk, Mark, et al. Capability Maturity Model: Guidelines for Improving the Software Process. Boston: Addison-Wesley, June 1995.
2. Software Engineering Institute. Capability Maturity Model® Integrated℠. Pittsburgh: SEI.
3. Harry, Mikel J., and Richard Schroeder. Six Sigma: The Breakthrough Management Strategy Revolutionizing The World's Top Corporations. New York: Doubleday, Dec. 1999.
4. ISO/IEC Standard 9126. "Information Technology – Software Quality, Part 1." 1995.
5. Card, David, and William Agresti.

"Resolving the Software Science Anomaly." Journal of Systems and Software Vol. 7 (1990): 29-35.
6. Card, David. "Quantitatively Managing the Object-Oriented Design Process." Canadian National Research Council Conference on Quality Assurance of Object-Oriented Software. Feb. 2000.
7. Gaffney, John. "Some Models for Software Defect Analysis." Lockheed Martin Software Engineering Workshop, Gaithersburg, MD, Nov. 1996.
8. Software Productivity Consortium. SWEEP Users Guide. SPC-98030-MC, 1997.
9. Harbaugh, Sam. "Crusader Software Quality Assurance Process Improvement." Technical Report. Integrated Software, Inc., 2002.
10. Hays, William, and Robert Walker. Statistics: Probability, Inference, and Decision. Austin, TX: Holt, Rinehart, and Winston, 1970.
11. Card, David. "Learning From Our Mistakes With Defect Causal Analysis." IEEE Software Jan. 1998.
12. Chillarge, R., et al. "Orthogonal Defect Classification." IEEE Transactions on Software Engineering Nov. 1992.

Note

1. An earlier version of this article was published in the proceedings of the Institute of Electrical and Electronics Engineers' Computer Software and Applications Conference, Aug. 2002.

About the Author

David N. Card is a fellow of the Software Productivity Consortium where he provides technical leadership in software measurement and process improvement. During 15 years at Computer Sciences Corporation, Card spent six years as the director of Software Process and Measurement, one year as a resident affiliate at the Software Engineering Institute, and seven years with the research team supporting the NASA Software Engineering Laboratory. Card is editor-in-chief of the *Journal of Systems and Software*. He is the author of "Measuring Software Design Quality," co-author of "Practical Software Measurement," and co-editor of ISO/IEC standard 15939:2002 "Software Measurement Process." Card is a senior member of the American Society for Quality.

Software Productivity Consortium
2214 Rock Hill Road
Herndon, VA 20170
Phone: (703) 742-7199
Fax: (703) 742-7200
E-mail: card@software.org

WEB SITES

Software Quality HotList

www.soft.com/Institute/HotList

The Software Research Institute maintains a list of links to selected organizations and institutions that support the software quality and software testing area. Organizations and other references are classified by type, by geographic area, and then in alphabetic order within each geographic area. The institute's aim is to bring to one location a complete list of technical, organizational, and related resources.

The Quality Assurance Institute

www.qaiusa.com

The Quality Assurance Institute (QAI) is exclusively dedicated to partnering with the enterprise-wide information quality profession. QAI is an international organization consisting of member companies in search of effective methods for detection-software quality control and prevention-software quality assurance. QAI provides consulting, education services, and assessments.

Software Technology Support Center

www.stsc.hill.af.mil

The Software Technology Support Center is an Air Force organization established to help other U.S. government organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

COMING EVENTS

March 24-28
International Symposium on Integrated Network Management
Colorado Springs, CO
www.im2003.org

March 31-April 2
Association for Configuration and Data Management's Annual Technical and Training Conference
San Diego, CA
www.acdm.org/main.htm

April 1-2
SecurE-Biz Summit
Arlington, VA
www.SecurE-Biz.net

April 8-10
FOSE 2003
(Federal Office Systems Exposition)
Washington, D.C.
www.fose.com

April 28-May 1
Software Technology Conference 2003

Salt Lake City, UT
www.stc-online.org

May 3-10
International Conference on Software Engineering
Portland, OR
www.icse-conferences.org/2003

May 12-16
STAREAST '03
Orlando, FL
www.sqe.com/stareast/

June 2-6
Applications of Software Measurement
San Jose, CA
www.sqe.com/asm

August 19-22
Software Test Automation Fall '03
Boston, MA
www.sqe.com/testautomation/