# Predicting Software Errors and Defects

by

Mark Criscione, Motorola

Jim Ferree, Motorola

Don Porter, Motorola

**ABSTRACT**

*Phase Containment Effectiveness (PCE) measures the ability of software development phases to detect faults. A weakness of this measure is that it determines how effective the phase WAS. This paper presents a fault prediction model that uses the PCE and related process measures to statistically predict the number of:*

- *Faults = mistakes introduced into design documents or software code.*
- *Errors = faults detected within the current phase.*
- *Defects = faults that escape the current phase.*

*The model also predicts customer found defects.*

*To create the model, means ($\mu$) and standard deviations ($\sigma$) are derived from historical data for the following process measures:*

- *Size = lines of code or pages of documentation.*
- *Fault Density = Faults/Size.*
- *Phase Containment Effectiveness = Errors/Faults.*
- *Phase Screening Effectiveness (PSE) = (Defects detected / Defects escaping prior phases).*

*For new software releases, these measures are used as spreadsheet inputs. Model outputs include predictions and confidence limits for faults, errors and defects, by phase and total.*

*Template spreadsheets are provided to project managers. The output cells contain formulae that embody expressions for output means and standard deviations, which are products and quotients of input variables. Because these formulae are quite complex, we used Monte-Carlo simulations to verify spreadsheet accuracy.*

*Software development is guided by comparing actual to predicted values using "stop light" tables. A "green light" indicates that the actual value is in the 68% confidence interval ($\mu \pm \sigma$), yellow is used for the 95% confidence interval ($\mu \pm 2\sigma$). Values outside the latter interval are painted red and considered to be "out of control". Layer charts, with similar coloring, compare actual to predicted values for each measure and phase.*

*As development proceeds, project managers can replace previously estimated quantities with their realized values. This provides the twin benefits of reducing bias for future estimates, and narrowing confidence limits by shrinking the related standard deviations.*

*The innovations in our work include:*

- *Proactive use of metrics that had previously been used retroactively.*
- *Careful use of probability theory to provide accurate predictions.*
- *Monte Carlo simulation techniques for model validation.*

**KEY WORDS**

*Software Inspection, Defect Prevention, Fault Prediction, Probability Model*

## THE PROBLEM

The estimated cost to fix a software defect after release to the customer is 100 times greater than if it is found in the software development phase of origination. This cost may range from $8,000 to $40,000, depending on the severity and size of the defect.[1] In addition to these direct costs, there is the cost of lost customers.

In an attempt to control defects, our organization has calculated "Phase Containment Effectiveness" (PCE) metrics for years. These metrics indicate how well people in the several software life cycle phases capture the mistakes that they introduce. Unfortunately, the procedure suffered various drawbacks, including:

- Focus. The focus was on metrics that cannot be accurately calculated until after the software is released. This is too late in the software life cycle to allow for proactive actions.
- Lack of prediction. There was never an attempt to inform project managers whether they were capturing a reasonable amount of errors. Lacking such predictions and associated confidence intervals, it was impossible for the managers to pro-actively improve the quality of their phase's work.
- PCEs were produced individually for each phase, without a model that covered the entire life cycle.

## THE SOLUTION

Our solution to these problems was to develop a comprehensive model for predicting the number of errors expected to be produced each phase, as well as the number of errors that should be captured. The model uses applied probability to provide confidence intervals, so that the project manager knows, in real time, when his or her group has produced, or captured, an unusually high or low number of errors.

The tool is embodied in an Excel spreadsheet with input parameters provided by historical data. Industry standard values may be used if no historical data is available. However, the managers can play out "what-if" scenarios by changing these inputs to compensate for any process changes. Variations on the spreadsheets allow the managers to employ different modeling strategies, depending on whether they are more confident in predicted error introduction rates or error capture rates.

The tool also allows the user to input actual data as it becomes available. By replacing estimates with actual data, subsequent predictions become more accurate and more precise. Graphical and tabular analyses are included. These allow comparison of actual outcomes to predicted. The graphs provide green, yellow and red zones. The green zone shows that actual outcomes are consistent with predictions, yellow is a warning zone, and the red zone tells the project manager that the actual errors are inconsistent with predictions.

## BENEFITS

The following are among the benefits achieved:

- The different phases of development are now connected. If much fewer errors are found in phase k than expected, the managers in phase k+1 will be on the lookout for those which escaped capture.
- The template provides a common PCE approach that all groups in our organization can employ. This enhances communication.
- The model generates estimates of faults that testing groups can expect to find, with 95% confidence intervals.
- We can predict the number of faults that our customers can expect to find.

## BASIC FAULT MODEL

There is a fundamental identity linking faults (mistakes introduced into design documents or software code), errors (faults detected within the current phase), and defects (faults that escape the current phase):

**Faults ≡ Errors + Defects**

Depending on the specific need, a prediction model can be developed to predict errors, faults or defects. In the model presented in this paper, both errors and defects are predicted.

The fault prediction model is based on software lifecycle phases. The phases we used are the following:

- Requirements - Describes in abstract terms what the software is to accomplish.
- Design - Provides a detailed description of software functionality, including pseudo-code.
- Code - The actual writing of the software.
- Development Test - Tests individual features in a software package.
- System Test - Tests the entire software package.
- Post Release - Like a "Beta test", tests the software package in its first field release.
- Customer - This phase refers to the software package after release to the field.

The model uses historical data collected from similar software releases to predict fault creation and detection in future releases. The historical inputs to the model are Fault Density (FD), Phase Containment Effectiveness (PCE) and Phase Screening Effectiveness (PSE). Another input is the estimated size (S) of the future release.

The fault density is calculated for each creation phase (requirements, design, and code) which is the total number of faults for the phase divided by the size of the phase. The size is measured as the number of pages inspected for both requirements and design, and lines of code created for the code phase.

The phase containment effectiveness is also calculated for each creation phase. The PCE is the number of errors found during inspection divided by the total faults for each phase. This gives an indication of the capability of the process to capture faults.

The phase screening effectiveness measures the ability of a phase to capture defects escaped from prior phases. There is no phase prior to requirements and thus it has no PSE. The design PSE equals the number of defects captured during the design phase divided by the defects escaped from the requirements phase. The code PSE equals the defects captured during the coding phase divided by the defects escaped from prior phases (design defects plus any remaining requirements defects). The PSE for development test is the number of defects captured during development test divided by the number of defects entering that phase. The same logic applies to the system test phase and the post release phase. Since the customer phase is last, no defects can escape. Therefore, PSE is defined to equal 100%.

## Data Confidentiality

The data presented is not actual Motorola data. The authors chose to show data that best illustrates fault model concepts while maintaining the confidentiality of Motorola data.

## Historical Data

Data was collected from three prior releases of software. For each input parameter, the average is represented as a pooled average. The release to release variation is represented by the standard deviation. The historical averages and standard deviations are used to predict a future release named "A".

The estimates for the size of release A are shown in Table 1. The number of faults expected for each creation phase is the product of the estimated size and the fault density as shown in Table 1.

**Table 1: Estimated Faults**

| Phase | Size | Fault Density | Faults |
|---|---|---|---|
| Requirements | 250 | 0.400 | 100 |
| Design | 900 | 0.300 | 270 |
| Code | 40000 | 0.015 | 600 |

The phase containment effectiveness determines what percentage of these faults will be captured as errors. The number of errors is the product of the number of faults and the PCE. The remainder of the faults are defects. Table 2 shows the errors and defects from the estimated faults in Table 1.

**Table 2: Estimated Errors and Defects**

| Phase | Faults | PCE | Errors | Defects |
|---|---|---|---|---|
| Requirements | 100 | 70% | 70 | 30 |
| Design | 270 | 80% | 216 | 54 |
| Code | 600 | 90% | 540 | 60 |

We have now calculated the number of errors that we expect to capture for each phase and the number of defects that will escape, but we do not know in which phase the defects will be captured. The Phase Screening Effectiveness will answer this. The expected number of defects captured in a phase equals the PSE times the number of defects escaping from prior phases. In Table 3 we will use the 30 requirements defects from Table 2 to illustrate where these defects will be captured. From the design PSE of 10% we expect to capture 30 * 10% = 3 requirements defects during the design phase leaving 27 defects remaining. The code PSE of 20% applied to the 27 remaining defects will yield an estimated 5.4 requirements defects leaving 21.6 remaining. Continuing toward the bottom of the table, the post release PSE of 60% will capture 5.5 requirements defects leaving 3.6 to be detected by the customer.

**Table 3: Estimated Requirement Defects by Phase Captured**

| Phase | Defects Entering the Phase | PSE | Req. Defects Captured | Defects Remaining |
|---|---|---|---|---|
| Requirements | - | - | - | 30 |
| Design | 30.0 | 10% | 3.0 | 27.0 |
| Code | 27.0 | 20% | 5.4 | 21.6 |
| Dev. Test | 21.6 | 30% | 6.5 | 15.1 |
| System Test | 15.1 | 40% | 6.0 | 9.1 |
| Post Release | 9.1 | 60% | 5.5 | 3.6 |
| Customer | 3.6 | 100% | 3.6 | - |

By applying the same logic to 54 design defects and 60 code defects from Table 2 and summing the defects captured in each phase we can estimate the defects captured by each phase as shown in Table 4.

**Table 4: Estimated Defects Captured by Phase**

| Phase | Req. | Design | Code | Total |
|---|---|---|---|---|
| Requirements | - | - | - | - |
| Design | 3.0 | - | - | 3.0 |
| Code | 5.4 | 10.8 | - | 16.2 |
| Dev. Test | 6.5 | 12.9 | 18 | 37.4 |
| System Test | 6.0 | 12.1 | 16.8 | 34.9 |
| Post Release | 5.5 | 10.9 | 15.1 | 31.5 |
| Customer | 3.6 | 7.3 | 10.1 | 21.0 |
| **Total** | 30.0 | 54.0 | 60.0 | 144.0 |

## FAULT MODEL WITH SIGMA LIMITS

From the Table 4 we calculated the expected number of defects captured for each phase. But how confident are we in this number?

To gain confidence in our expected values we calculated the standard deviations ($\sigma$) of the historical data. With this information we created limits of one and two standard deviations above and below the expected values.

Table 5 shows the expected value (most likely), the standard deviation, and the upper and lower limits for fault density. From this data we are approximately 68% confident that we are within the upper and lower one sigma limits, and 95% confident that the fault density will fall between the upper and lower two sigma limits. Those interested in the formulae for these predictions and confidence limits may refer to Appendix A.

**Table 5: Estimated Fault Density with Sigma Limits**

| Phase | Sigma | Two Sigma Lower Limit | One Sigma Lower Limit | Most Likely | One Sigma Upper Limit | Two Sigma Upper Limit |
|---|---|---|---|---|---|---|
| Req. | 0.1 | 0.200 | 0.300 | 0.400 | 0.500 | 0.600 |
| Design | 0.075 | 0.150 | 0.225 | 0.300 | 0.375 | 0.450 |
| Code | 0.002 | 0.011 | 0.013 | 0.015 | 0.017 | 0.019 |

By applying the same logic we used earlier to calculate the number of faults in Table 1, we calculate upper and lower limits for the expected number of faults as shown in Table 6.

**Table 6: Estimated Faults with Sigma Limits**

| Phase | Sigma | Two Sigma Lower Limit | One Sigma Lower Limit | Most Likely | One Sigma Upper Limit | Two Sigma Upper Limit |
|---|---|---|---|---|---|---|
| Req. | 26.9 | 46.1 | 73.1 | 100.0 | 126.9 | 153.9 |
| Design | 72.7 | 124.6 | 197.3 | 270.0 | 342.7 | 415.4 |
| Code | 100.0 | 400.0 | 500.0 | 600.0 | 700.0 | 800.0 |

The same concept of upper and lower limits is applied to PCE to give us confidence intervals for captured errors. Table 7 shows the estimated errors by creation phase with upper and lower limits.

**Table 7: Estimated Errors with Sigma Limits**

| Phase | Sigma | Two Sigma Lower Limit | One Sigma Lower Limit | Most Likely | One Sigma Upper Limit | Two Sigma Upper Limit |
|---|---|---|---|---|---|---|
| Req. | 24.1 | 21.8 | 45.9 | 70.0 | 94.1 | 118.2 |
| Design | 64.1 | 87.8 | 151.9 | 216.0 | 280.1 | 344.2 |
| Code | 94.9 | 350.3 | 445.1 | 540.0 | 634.9 | 729.7 |

Applying this concept to PSE gives us confidence intervals for defects. Table 8 shows the estimated defects captured by phase with upper and lower limits.

**Table 8: Estimated Defects Captured by Phase with Sigma Limits**

| Phase | Sigma | Two Sigma Lower Limit | One Sigma Lower Limit | Most Likely | One Sigma Upper Limit | Two Sigma Upper Limit |
|---|---|---|---|---|---|---|
| Design | 3.2 | 0.0 | 0.0 | 3.0 | 6.2 | 9.5 |
| Code | 17.7 | 0.0 | 0.0 | 16.2 | 33.9 | 51.7 |
| Dev Test | 23.2 | 0.0 | 14.2 | 37.4 | 60.7 | 83.9 |
| System Test | 30.1 | 0.0 | 4.8 | 34.9 | 65.1 | 95.2 |
| Post Release | 25.9 | 0.0 | 5.5 | 31.4 | 57.4 | 83.3 |
| Custom er | 20.9 | 0.0 | 0.1 | 21.0 | 41.9 | 62.7 |

## ANALYSIS

A stop light table shown in Table 9 compares actual results to predictions. Green indicates that the phase is within the one sigma limits (68% confidence interval) and Yellow extends to the two sigma limits (95% confidence interval). Values outside the latter interval are Red and considered to be "out of control".

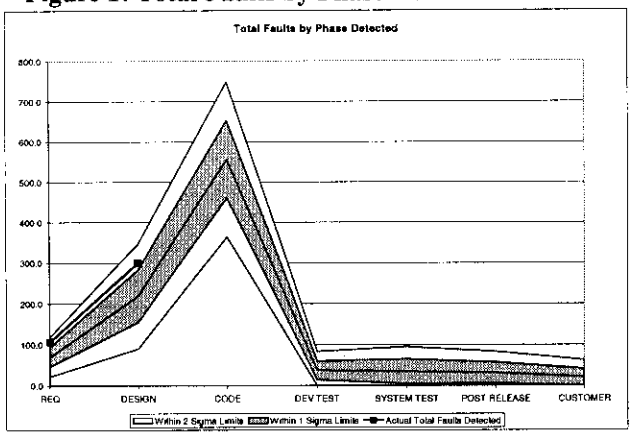
**Table 9: Stop Light Table for Actual Data**

| Phase | Actual Size | Actual Fault Density | Actual Faults | Actual PCE % | Actual Errors | Actual Defects |
|---|---|---|---|---|---|---|
| REQ | 356 | 0.4466 | 159 | 67.3% | 107 | 52 |
| DESIGN | 1138 | 0.3243 | 369 | 77.2% | 285 | 84 |
| CODE | 38864 | 0.0170 | 662 | 87.9% | 582 | 80 |

The actual size for the requirements and design phases is red because it is above the two sigma limit. This translates into increased number of actual faults for requirements and design.

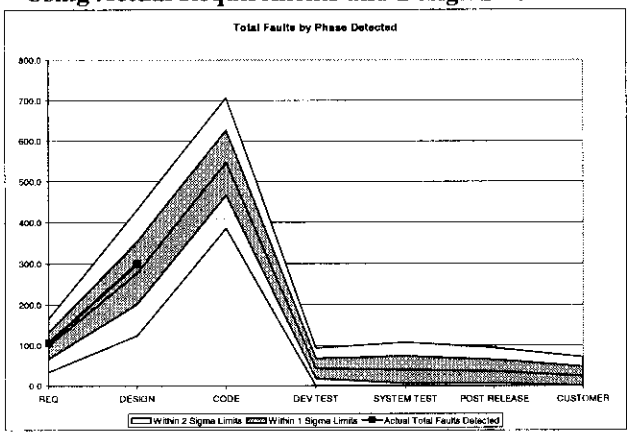
Figure 1 shows the total faults by phase detected as a layer chart. The middle shaded area denotes the upper and lower one sigma limits and the outer shaded area indicates the two sigma limits. The actual faults detected line is in the upper two sigma area due to the larger than expected actual size noted above.

**Figure 1: Total Faults by Phase Detected**




To improve the estimate we provided an option in the spreadsheet to substitute actual size for the estimated size and reduce the standard deviations to 25% of their initial values. Figure 2 shows the effect of adjusting for actual requirements and design size. Notice the actual faults are now within the one sigma limits. Also, there is a slight increase of expected faults in later phases due to the size increase.

**Figure 2: Total Faults by Phase Detected Using Actual Requirements and Design Size**




We also provided an option in the spreadsheet to substitute actual errors for the completed phases and reduce the standard deviations to 25% of their initial values. This correction is applied when few additional inspections are expected. The standard deviations do not disappear because experience has shown that additional errors may be detected (and even introduced) when older errors are corrected. Historical data supports the 25% value. We typically adjust for actual errors when the number of actual errors is near or above the estimated number of faults (implying few or negative defects). The new estimate for errors is the actual errors detected and the new estimate for faults is the actual errors divided by the PCE.

Figure 3 shows the effects of adjusting for actual errors found in requirements and design. It also shows actual faults for later phases. Notice how the confidence limits shrink for the requirements and design phases.

**Figure 3: Total Faults by Phase Detected Using Actual Requirements and Design Errors**

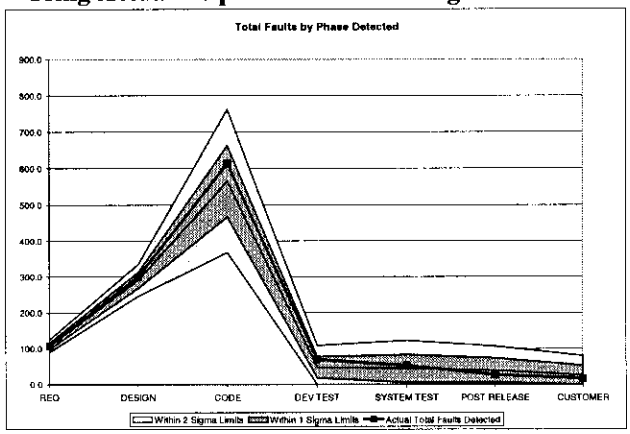



Table 8 shows an initial prediction of 21 customer defects. After adjusting for actual size and error data, the expected customer defects rose to 27. Also, the upper 95% confidence limit increased from 63 to 81. The actual 18 customer defects detected were within the 68% confidence intervals.

## RESULTS

One way to measure the accuracy of the fault prediction model is to compare actual customer defects to predicted customer defects. Table 9 shows the number of standard deviations that the actual customer defects are from the predicted customer defects. The calculation used is:

Number of Std. Dev. = (Actual - Predicted) / Sigma

The releases are shown in columns and rows represent results with no adjustments, adjusting for size only, and adjusting for size and actual errors. Notice that on average, the accuracy improves as size is adjusted and slightly more when size and errors are adjusted.

**Table 9: Number of Standard Deviations that Actual Customer Defects are from Predicted Customer Defects**

| Model Adjustments | Rel. A | Rel. B | Rel. C | Rel. D | Avg. |
|---|---|---|---|---|---|
| None | 2.30 | 0.47 | 0.46 | 0.99 | 1.06 |
| Size | 0.68 | 0.75 | 0.76 | 1.07 | 0.82 |
| Size, Errors | 0.19 | 0.00 | 0.93 | 1.04 | 0.54 |

Groups using the fault model also observed a greater understanding of their data and awareness of faults remaining to be detected.

## SIMULATION

The mathematical formulae for the fault prediction models are somewhat difficult. This is especially true when entering them as cell formulae in a spreadsheet, so it is very easy to make mistakes. Simulations have the distinct advantage of not requiring complicated calculations. So we ran Monte Carlo simulations to check our work.

In the simulations, normal probability distributions were specified for each input parameter. Even though the input parameters possess other distributions, sample sizes are large enough to justify normal approximations. For each simulation iteration, values were randomly selected from these input distributions and used to calculate output values (sizes, fault densities, phase containment effectiveness and phase screening effectiveness). After 500 iterations were run, means and standard deviations of the output variables were compared to the outputs in the spreadsheet. This proved to be a valuable exercise, because it pointed to a few errors that substantially inflated the prediction intervals.

In the future, we plan to increase the realism of the model. As we move in that direction, Monte Carlo simulation will become an increasingly valuable tool for checking spreadsheet accuracy.

## CONCLUSION

We introduced a fault model that predicts the number of errors and defects throughout the development cycle. Project managers can use this information to quantitatively determine if the development process is in control (green condition), may be going out of control (yellow condition) or is clearly out of control (red condition). This model is able to adjust estimates based on the most current data available. It was applied to predict Release A errors and defects. At time of publication, 18 customer problems were observed which is within the 68% confidence limits.

## FUTURE DIRECTIONS

There are several opportunities for enhancing the model. One of these is to improve the initial estimates of input parameters. We would like to develop a model that can use information from the earliest stages of concept exploration and from other sources to predict likely input parameters for each release.

# APPENDIX A: MATHEMATICAL MODELS for FAULT PREDICTION

Statistical Background

The fault prediction model uses some basic results from probability theory concerning sums, differences, products and quotients of random variables. These results are described here.

Specifically, we used results that derive the mean and standard deviation of functions of a set of random variables. The mean of a random variable is the value it is expected to assume on average. For populations with a finite number (N) of values, the population **mean** is given by the expression:

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} X_i$$

As the mean is a measure of average location, the population variance is a measure of dispersion about that location. For a finite population, the **variance** is given by:

$$\sigma_x^2 = \frac{1}{N}\sum_{i=1}^{N} (X_i - \mu_x)^2$$

Notice that, like the mean, the variance is really an average. It is the average squared distance that individual members of the X population fall from the mean. While this if very useful mathematically, it suffers the drawback of being expressed in squared units (like dollars squared), and therefore may not be easily understood. To get around this problem, the square root, called the **standard deviation**, is often used in reports.

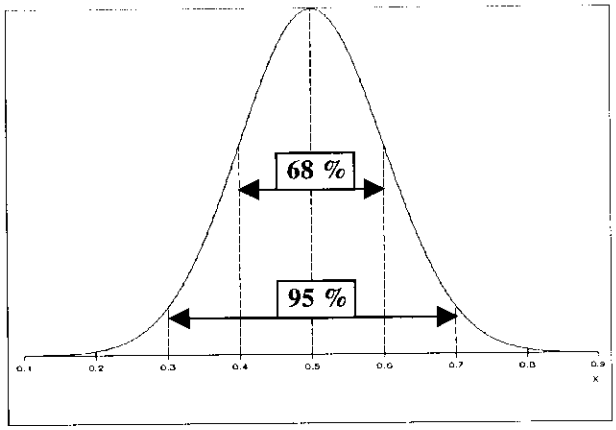
$$\sigma_x = \sqrt{\sigma_x^2}$$

It is convenient (despite a small mathematical nicety) to think of a standard deviation as the average distance that values of the random variable fall from the mean. Thus it serves as an excellent measure of dispersion. The smaller the standard deviation, the more crowded about the mean the values of X will be. Thus a smaller standard deviation implies a more stable process.

The mean and standard deviation of a random variable can be combined to form prediction intervals. If a random variable X possesses a reasonably symmetric, bell-shaped (normal) distribution, then regions defined by the mean and standard deviation will contain known percentages of all values. Such a probability distribution, with mean .5 and standard deviation .1, is depicted in figure 1. The region within one standard deviation of the mean contains approximately 2/3 of the values, a 2 standard deviation interval contains approximately 95% of the values, and the interval within three standard deviations contains almost all. That is:

- $\text{Prob}(\mu_x - \sigma_x \le X \le \mu_x + \sigma_x) \approx .68$
- $\text{Prob}(\mu_x - 2\sigma_x \le X \le \mu_x + 2\sigma_x) \approx .95$
- $\text{Prob}(\mu_x - 3\sigma_x \le X \le \mu_x + 3\sigma_x) \approx .997$

**Figure 1: A Normal Distribution**




The first two intervals define zones in the fault prediction model. If the observed number of errors falls in the first prediction interval, everything is proceeding as predicted. If it falls in the second interval, but outside the first, it is considered a "warning". When errors fall outside the second zone, the project manager should investigate to see why either more or less than the predicted number of errors are being found. If more, is it because of a bad product or an intensive search for errors? If less, is it because the product is of superior quality, or is it not being properly inspected?

Statistical Model of Fault Prediction

The modeling begins with the means and standard deviations of several estimated quantities:

- Size (S), whether in pages of documentation or lines of code, is estimated at the beginning of the development cycle. Of course, these estimates are subject to error. The mean and standard deviation of size estimates are denoted $\mu_s$ and $\sigma_s$.
- Fault density (FD), gives the *rate* that faults are introduced into a software product. Fault densities are recorded as faults per page of documentation or faults per line of code. The mean and standard deviation are denoted $\mu_{FD}$ and $\sigma_{FD}$. Alternate models use error densities instead. An "ED" subscript is used in this case. An "F" subscript denotes number of faults, a derived quantity.
- The phase containment effectiveness (PCE) measures the ability to capture a fault before it escapes to the next phase of software

development. For example, it may measure the ability of software designers to fix their mistakes before the beginning of the next phase, coding. The mean and standard deviations of PCEs are denoted $\mu_{PCE}$ and $\sigma_{PCE}$.

- When a fault is detected in the phase of origination, it is called an "error". If it escapes to subsequent phases, it is called a "defect". The Phase Screening Effectiveness (PSE) measures the ability of a given phase to capture the defects of prior phases. We denote the mean and standard deviations as $\mu_{PSE}$ and $\sigma_{PSE}$.

All of these input parameters obey known probability distributions, which are helpful in model building. We have found that there is a lot of variation in these values from release to release. So we used the standard deviations of realized values over past releases as the sigma inputs. For example, if we had five historical PCEs, then the PCE inputs were calculated by:

- $\mu_{PCE}$ = Total errors/total faults
- $\sigma_{PCE}$ = Standard deviation of the five individual PCEs.

The mean PCE is thus a pooled estimate of historical data, and the standard deviation is the between-release standard deviation. Statistically, this standard deviation works quite well because the component of variation attributable to releases tends to overwhelm the component due to uncertainty within a release.

**Predicting Faults**

At the beginning of development, nothing is known for sure about the size of the product, the rate at which faults are introduced, or the rates at which they are captured. However, we do have some historical evidence. So we begin with a size estimate and standard deviation, and likewise with estimates and standard deviations of fault densities and PCEs. For the Requirements phase, we predict the number of faults introduced by multiplying the estimated size by the estimated fault density (FD), as in equation 1A.

(1A) $F = S * FD$

When the project manager inputs the means and standard deviations of size and fault density, the spreadsheet automatically calculates the mean and standard deviation of the predicted number of faults, given by equations 1B and 1C, respectively.

(1B)

(1C)

The mean of a product of two random variables is better approximated as the product of the means plus their covariance. So expression 1B assumes that the size and fault density variables are statistically independent, since the covariance is then zero. This is one of the simplifying assumptions made for the spreadsheet model (which may be dropped for more realism, but only at the expense of additional complexity).

**Predicting Errors and Defects**

Expression 1B gives the expected number of faults, but does not determine how many of these faults will be captured in phase (thus becoming errors), and how many will float downstream as defects. To determine the number of errors, we input the mean and standard deviation of the PCE to derive total errors (E) for the phase. The calculations are given in equation 2A-2C. These equations are also shown in expanded form by substituting the calculation for faults:

(2A) $E = PCE * F = PCE * S * FD$

(2B) $\mu_E = \mu_{PCE} * \mu_F = \mu_{PCE} * \mu_S * \mu_{FD}$

(2C) $\sigma_E = \sqrt{\mu_{PCE}^2 * \sigma_F^2 + \sigma_{PCE}^2 * \mu_F^2} =$

$$\sqrt{\mu_{PCE}^2 * \mu_S^2 * \sigma_{FD}^2 + \mu_{PCE}^2 * \sigma_S^2 * \mu_{FD}^2 + \sigma_{PCE}^2 * \mu_S^2 * \mu_{FD}^2}$$

Similar expressions for total defects escaped (DE) are shown in equations 3A-3C.

(3A) $DE = (1 - PCE) * F = (1 - PCE) * S * FD$

(3B) $\mu_D = (1 - \mu_{PCE}) * \mu_F = (1 - \mu_{PCE}) * \mu_S * \mu_{FD}$

(3C) $\sigma_D = \sqrt{(1-\mu_{PCE})^2 * \sigma_F^2 + \sigma_{PCE}^2 * \mu_F^2} =$

$$\sqrt{(1-\mu_{PCE})^2 * \mu_S^2 * \sigma_{FD}^2 + (1-\mu_{PCE})^2 * \sigma_S^2 * \mu_{FD}^2 + \sigma_{PCE}^2 * \mu_S^2 * \mu_{FD}^2}$$

**Predicting Captured Defects**

The first phase in the software development cycle is requirements, so it has no Phase Screening Effectiveness. The next phase is design, which has expressions to explain its own faults, errors and defects. But design, and subsequent phases, also have PSE values. The predicted number of defects captured (DC) is the product of the estimated number of defects escaped (DE) from previous phases and the PSE. This is given by expression 4A. Because these expressions span consecutive software development phases, superscripts are used to denote the phase of origin. Thus R = Requirements, D = Design, C = Code, and so forth.

(4A) $DC^D = PSE^D * DE^R$, where

$DC^D$ = Defects Captured in Design, and

$DE^R$ = Defects Escaped from Requirements

The mean and standard deviation of faults captured are given in equations 4B and 4C. Symbols inside parentheses indicate phase.

(4B)

(4C)

Expressions like 4B and 4C get pretty scary for later phases. For example, the predicted number of defects captured in the coding phase is obtained by multiplying its PSE times the predicted number of defects entering that phase. The latter includes not only the defects created in design, but the defects created in requirements that were not captured in design. For the coding phase, then, the prediction equation for defects captured is given in equation 5A, the mean and standard deviation are given in equations 5B - 5C.

(5A) $DC^C = PSE^C * \{(1 - PSE^D) * DE^R + DE^D\}$
$= PSE^C * \{DE^{R'} + DE^D\}$

where $DE^{R'}$ represents requirements phase defects that were not captured in the design phase.

(5B) $\mu_{DC}^{C} = \mu_{PSE}^{C} * \{\mu_{DE}^{R'} + \mu_{DE}^{D}\}$

(5C) $\sigma_{DC}^{C} = \sqrt{\mu_{PSE(C)}^{2} * [\sigma_{DE(R')}^{2} + \sigma_{DE(D)}^{2}] + \sigma_{PSE(C)}^{2} * [\mu_{DE}^{R'} + \mu_{DE}^{D}]^2}$

In 5C, certain subscripts are followed by parentheses that identify the phase to which the quantity belongs. For example, PS(C) identifies the PSE for the coding phase. The mean and variance of $DE^{R'}$ are given in 5D and 5E:

(5D) $\mu_{DE}^{R'} = (1 - \mu_{PSE}^{D}) * \mu_{DE}^{R}$

(5E) $\sigma_{DE(R')}^{2} = (1 - \mu_{PSE}^{D})^2 * \sigma_{DE(R)}^{2} + \sigma_{PSE(D)}^{2} * \mu_{DE(R)}^{2}$

The spreadsheet model breaks these calculations into smaller pieces, providing additional information on the predicted number of defects that are not detected in subsequent phases.

Several testing phases follow coding. No new defects are introduced during test, so the only quantities to calculate are phase screening effectiveness. Expressions like 5A-5E determine the number of defects captured in these phases, along with their respective standard deviations. These expressions are also quite complicated, since they contain elements for defects that have escaped all previous phases.

## REFERENCES

[1] "Software Engineering Economics", Barry W. Boehm, Prentice Hall, October 1981.

[2] "Mathematical Statistics and Data Analysis", John A. Rice, Wadsworth & Brooks/Cole, 1988

## Mark Criscione

Mark Criscione began working in the technical industry as an integrated circuit designer of computer keyboards and terminals for AT&T Teletype Corporation in 1984 after graduating from the University of Illinois at Chicago with a Bachelor's Degree in electrical engineering.

He later shifted his focus to software system testing and received a Master's Degree in electrical engineering and computer science from the University of Illinois at Chicago in 1990. His interest in metrics was initiated as he worked in the software reliability area predicting the number of system test faults expected. He joined Motorola in 1993 and developed his expertise in software metrics. He became a Six Sigma Black Belt which is an achievement in applying statistical methods, leadership, networking, and continuous improvement toward total customer satisfaction. Mark's current responsibility is the definition of corporate and project level metrics.