

Software quality control and prediction model

Norman F. Schneidewind

Norman F. Schneidewind, "Software quality control and prediction model for maintenance", Annals of Software Engineering, [Baltzer Science Publishers](#), Volume 9 (2000), May 2000, pp. 79-101

Division of Computer and Information Sciences and Operations

Naval Postgraduate School

2822 Racoon Trail

Pebble Beach, CA 93953

Voice: (831) 656-2719

Fax : (831) 372-0445

Email: nschneid@nps.navy.mil

Abstract

We develop a quality control and prediction model for improving the quality of software delivered by development to maintenance. This model identifies modules that require priority attention during development and maintenance by using Boolean discriminant functions. The model also predicts during development the quality that will be delivered to maintenance by using both point and confidence interval estimates of quality. We show that it is important to perform a marginal analysis when making a decision about how many metrics to include in a discriminant function. If many metrics are added at once, the contribution of individual metrics is obscured. Also, the marginal analysis provides an effective rule for deciding when to stop adding metrics. We also show that certain metrics are dominant in their effects on classifying quality and that additional metrics are not needed to increase the accuracy of classification. Related to this property of *dominance* is the property of *concordance*, which is the degree to which a set of metrics produces the same result in classifying software quality. A high value of *concordance* implies that additional metrics will not make a significant contribution to accurately classifying quality; hence, these metrics are redundant. Data from the *Space Shuttle* flight software are used to illustrate the model process.

1. Introduction

A key problem in maintenance is to identify problems in the software during development before it reaches maintenance. To this end, we develop a quality control and prediction model that is used to identify modules that require priority attention during development and maintenance. This is accomplished in two activities: *validation* and *application*. Both activities occur during software development. *Validation* is an activity that is required in order to identify metrics that can identify low quality software that requires corrective action. *Application* is an activity during which validated metrics are applied to control and predict software quality. During *validation*, we use a build of the software that has been developed as the source of data to compute a discriminant function (i.e., a statistical method that is used to classify software quality) that we use to retrospectively classify and predict quality with specified accuracy, by build and module. Using this discriminant function during *application*, we classify and predict the quality of new software that is being developed. We make both point and confidence interval estimates of quality. This is the quality we expect to experience during maintenance.

During *validation*, both quality factor (e.g., discrepancy reports of deviations between requirements and implementation) and software metrics (e.g., size, structural) data are available; during *application*, only the latter are available. During *validation*, we construct Boolean discriminant functions (BDFs) comprised of a set of metrics and their critical values (i.e., thresholds). A BDF is a Boolean function consisting of *AND* and *OR* operators, module metric values, and metric critical values that is used to classify the quality of software. A metric critical value is a value in the range of the metric, estimated by using the inverse of the *Kolmogorov-Smirnov distance* (to be explained) that provides a threshold between two levels (e.g., *high* and *low*) of the quality of the software. We select the best BDF based on its ability to achieve the maximum relative incremental quality/cost ratio. During *application*, if at least one of the module's metrics has a value that exceeds its critical value, the module is identified as "high priority" (i.e., low quality); otherwise, it is identified as "low priority" (i.e., high quality). Our objective is to identify and correct quality problems during development so that a high quality product can be delivered to maintenance, as opposed to waiting until maintenance when the cost of correction would be high.

We use nonparametric statistical methods to: 1) identify the critical values of the metrics and 2) find the optimal BDF based on its ability to satisfy both *statistical* and *application* criteria. Statistical criteria refer to the ability to correctly classify the software (i.e., classify high quality software as high quality and low quality software as low quality). Application criteria refer to the ability to achieve a high quality/cost ratio. A BDF compares a module's metric value with the metric's critical value, for a set of metrics, in classifying the quality of the software. The BDFs provide good accuracy (i.e., $\leq 3\%$ error) for classifying quality factors. These functions make fewer mistakes in classifying software that is low quality than is the case when linear vectors of metrics are used because the critical values provide additional information for discriminating quality. In addition, we develop an effective stopping rule for adding metrics to the BDF that is based on quality/cost considerations.

We show that it is important to perform a marginal analysis (i.e., identification of the incremental contribution of each metric to improving quality) when making a decision about how many metrics

to include in the discriminant function. If many metrics are added to the set at once, the contribution of individual metrics is obscured. Also, the marginal analysis provides an effective rule for deciding when to stop adding metrics. We also show that certain metrics are dominant in their effects on classifying quality for *Space Shuttle* software (i.e., dominant metrics make fewer mistakes in classifying metrics than non-dominant ones) and that additional metrics are not needed to accurately classify quality. Related to the property of *dominance* is the property of *concordance*, which is the degree to which a set of metrics produces the same result in classifying software quality. A high value of *concordance* implies that additional metrics will not make a significant contribution to accurately classifying quality; hence, these metrics are redundant.

The contributions of this research are the following: 1) both statistical and application criteria should be used to determine which metrics and how many metrics should be used to classify maintenance quality; 2) a marginal analysis should be performed on each metric to determine whether its addition will increase the quality/cost ratio; 3) the Boolean discriminant function (BDF) is a new type of discriminant for classifying maintenance quality; 4) our application of Kolmogorov-Smirnov (K-S) distance is a new way to determine a metric's critical value; and 5) we have developed a new stopping rule for adding metrics: the ratio of the relative improvement in quality to the relative increase in cost.

1.1. Related research

Our model is one of a class of models concerned with the classification of quality, sometimes referred to as the identification of fault-prone modules. Porter and Selby [1990] used classification trees to partition multiple metric value space so that a sequence of metrics and their critical values could be identified that were associated with either high quality or low quality software. This technique is closely related to our approach of identifying a set of metrics and their critical values that will satisfy quality and cost criteria. However, we use statistical analysis to make the identification.

Briand et al. [1997] used logistic regression to classify modules as fault-prone or not fault-prone as a function of various object-oriented metrics. In another example of logistic regression, Khoshgoftaar and Allen [1997] used it to classify modules as fault-prone or not fault-prone as a function of faults, requirements, performance, and documentation software trouble report metrics. While one of our objectives is similar -- classify modules as either high quality or low quality -- we derive from this *binary* classification several predictive *continuous* quality and cost metrics. These metrics are used to predict the quality of software that will be delivered by development to maintenance and the cost of achieving it.

Khoshgoftaar et al. [1996a] used nonparametric discriminant analysis in each iteration of their military system project to predict fault-prone modules in the next iteration. This approach provided an advance indication of reliability and the risk of implementing the next iteration. They also conducted a similar study involving a telecommunications application, again using nonparametric discriminant analysis, to classify modules as either fault-prone or not fault-prone [Khoshgoftaar et al. [1996b]. Our approach has the same objective but we produce BDFs in terms of the original metrics as opposed to using density functions as discriminators.

Khoshgoftaar and Allen [1998] have also developed models for ranking modules for reliability improvement according to their degree of fault-proneness as opposed to whether they are fault-prone or not. They used Alberg Diagrams [Ohlsson and Alberg 1996] that predict percentage of faults as a function of percentage of modules by ordering modules in decreasing order of faults and noting the cumulative number of faults corresponding to various percentages of modules. The imperative in safety critical systems like the *Space Shuttle* is to investigate *all* suspect modules because even the module with the lowest a priori reliability risk could pose a safety hazard in operation. Our previous research showed a very high association between module failures and metric values that exceeded the critical values [Schneidewind 1995], as we will show later.

The following topics are covered: *Discriminative Power* model, approach to validation, and quality control and prediction applications of the model, Section 2; detailed description of validation methodology, Section 3; comparison of validation with application results for quality control and prediction, Section 4; quality point and confidence interval estimates, Section 5; comparison of BDF and linear discriminant function quality classification results, Section 6; development metric characteristics of modules that failed during maintenance, Section 7; and conclusions about the contributions of the model to quality control and prediction and the results obtained to date in applying it to the *Space Shuttle*, Section 8.

2. Discriminative power model

2.1. Discriminative power validation

Using our metrics validation methodology [IEEE 1998; Schneidewind 1992], and the *Space Shuttle* flight software metrics and discrepancy reports (DRs), we validate metrics with respect to the quality factor *drcount*. This is the number of discrepancy reports written against a module. In brief, this involves conducting statistical tests to determine whether there is a high degree of association between *drcount* and candidate metrics. As shown in Figure 1, we validate metrics on one random sample (Validation Sample) of 100 modules from Build 1 and apply the validated metrics to three random samples (Application Samples) of 100 modules each from Build 2 that are both disjoint among themselves and from the Validation Sample, drawn from a population of 1397 modules of *Space Shuttle* flight software. Nikora and Munson argue for the need of a measurement baseline against which evolving systems may be compared [Nikora and Munson 1998]. Our baseline is Build 1 in Figure 1. The measurement results from Build 1 provide the data source for controlling and predicting the quality delivered to maintenance and for comparing predicted with actual quality, once the latter is known. Next, we define *Discriminative Power*.

2.1.1. Discriminative power

Given the elements M_{ij} of a matrix of \mathbf{n} modules and \mathbf{m} metrics (i.e., \mathbf{nm} metric values), the elements MC_j of a vector of \mathbf{m} metric critical values, the elements F_i of a vector of \mathbf{n} quality factor values, and scalar FC of quality factor critical value, M_{ij} must be able to discriminate with respect to F_i , for a specified FC , as shown in the following relation:

$$M_{ij} > MC_j \Leftrightarrow F_i > FC \text{ and} \tag{1}$$

$$M_{ij} \leq MC_j \Leftrightarrow F_i \leq FC$$

for $i=1,2,\dots,n$, and $j=1,2,\dots,m$ with specified α , where α is the significance level of various statistical tests that are used for estimating the degree to which a set of metrics can correctly classify software quality. In other words, do the indicated metric relations imply corresponding quality factor relations in (1)? This criterion assesses whether MC_j has sufficient *Discriminative Power* to be capable of distinguishing a set of high quality modules from a set of low quality modules. If this is the case, we use the critical values in Quality Control and Prediction described below. The validation process is illustrated in Figure 1, where the critical values MC_j are produced in the Test phase of Build 1 by using the metrics M_{ij} from the Design phase and the quality factor F_i (e.g., *drcount*) that is available in the Test phase. Discrepancy reports are written against the software throughout development but they are not significantly complete until the end of the Test phase for a build during which failures are observed. The counts of discrepancy reports and metrics that are associated with a module were collected at the completion of a build by a metrics analyzer, using the source code as input. If a discrepancy report involves multiple modules, it is counted against every module affected. The desired quality level is set by the choice of FC . The lower its value, the higher the quality requirement; conversely, the higher its value, the lower the requirement. A value of zero is appropriate for safety-critical systems like the *Space Shuttle*.

It is important to recognize that validation is performed retrospectively. That is, with both metrics M_{ij} and quality factor F_i in hand for Build 1, we can evaluate how well the metrics would have performed if they had been applied to Build 1. If the metrics perform well, we say they are validated and it is our expectation that they will perform adequately when applied to Build 2. (i.e., not as well as when applied to Build 1 because of possible differences in module characteristics between Build 1 and Build 2 but better than using unvalidated metrics). Next, we describe the application of the model to quality control and prediction.

2.1.2. *Quality control and prediction*

Quality control is the evaluation of modules with respect to predetermined critical values of metrics. The purpose of quality control is to allow software managers to identify software that does not meet quality requirements early in the development process so corrective action can be taken when the cost is low. Quality control is applied during the Design phase of Build 2 in Figure 1 to flag modules below quality limits for detailed inspection. The validated BDFs, comprised of the metrics M_{ij} and their critical values MC_j that are obtained from Build 1, are used to either accept or reject the modules of Build 2 [Schneidewind 1997a; Schneidewind 1997b]. At this point in the development of Build 2, only the metric data M_{ij} and MC_j are available.

Quality predictions are used by the developer and maintainer to anticipate rather than react to quality problems. The predictions provide indications of the quality of the software that would be delivered to maintenance. Figure 1 shows the metrics controlling and predicting the quality of software that will be delivered to maintenance *early* in the development of Build 2. Accompanied by rigorous inspection and test, this process will result in improved quality of Build 2 and the software that is released to maintenance, of which Build 2 is a part. Once all of the quality factor data F_i (e.g., *drcount*) have been collected for Build 2, at the end of the Test phase as shown in Figure 1, the quality of Build 2 would be known. This, then, becomes the actual quality of Build 2 in the

maintained software.

3. Validation methodology

The basis of this model is a methodology for validating BDFs and their critical values that have the ability to discriminate high quality from low quality. We use a three stage process for selecting metrics for quality control and prediction: 1) compute critical values of the candidate metrics; 2) for the set of candidate metrics and critical values, find the optimal combination based on statistical and application criteria; and 3) apply a stopping rule for adding metrics. Table 1 provides a functional description of each stage. The three stages take place during the Test Phase of Build 1 of Figure 1, once all the quality factor data F_i (e.g., *drcount*) are available. The sections that follow provide the details of the statistical analysis for each stage.

Table 1. Functional Description of Metrics Validation Process			
	Statistical Test/ Procedure	Purpose	Result
Stage 1	Kolmogorov-Smirnov(K-S)	Compute the critical values of the candidate metrics.	Metrics ranked by K-S test results for input to Stage 2.
Stage 2	Contingency Table Analysis	Use the critical values obtained from Stage 1 to form a set of BDFs. Use the BDFs to estimate quality and cost of inspection for each set of metrics, starting with one metric, and increasing by one until the stopping rule is satisfied.	Metric sets with increasing numbers of metrics, each set with estimated quality and cost of inspection.
Stage 3	Stopping Rule for Adding Metrics	Add metrics to Stage 2 until the ratio of relative incremental quality to relative incremental inspection cost reaches a maximum.	Validated BDFs and their critical values that provide the highest estimated quality relative to the estimated cost of inspection.

3.1. Stage 1: compute critical values

Critical values MC_j are computed, using a new method we have developed, which is based on the Kolmogorov-Smirnov (K-S) test [Conover 1971]. This test was investigated for application to software metrics because of its ability to indicate the value of a metric (i.e., critical value) where maximum discrimination occurs between two samples of modules -- one of high quality and the other of low quality. The method has consistently yielded good results for controlling the quality of *Space Shuttle* software as our results will show. The K-S test is exact for continuous distributions and conservative (i.e., the true alpha is less than the specified value) for discrete metrics data [Conover 1971]. In addition, the large range (e.g., 0 - 2316 for *prologue size*) and fine granularity (e.g., units of one for *prologue size*) of the metrics data approximate continuous distributions. Thus, the K-S test is appropriate for analyzing metrics data.

Table 2 shows the metric definitions, critical values MC_j , and K-S distances for four metrics of the Validation Sample. These metrics were selected for analysis based on their relatively high K-S distance compared to other metrics that had been collected on the *Space Shuttle*. The K-S method tests whether the sample cumulative distribution functions (CDF) are from the same or different populations. The test statistic is the maximum vertical difference between the CDFs of two samples (e.g., the CDFs of M_{ij} for $drcount \leq FC$ and $drcount > FC$). If the difference is significant (i.e., $\alpha \leq .005$), the value of M_{ij} corresponding to maximum CDF difference is used for MC_j . This relationship is expressed in equation (2). This concept is illustrated in Figure 2, for the critical value of *prologue size*, where we show the CDFs for $drcount=0$ and $drcount>0$. In this example, the critical value is 38. This is the value of *prologue size* where there is the maximum difference between the CDFs. This is the value of *prologue size* where there is the maximum discrimination between high quality ($drcount=0$ curve) and low quality ($drcount>0$ curve). Metrics are added to the BDF in the order of their decreasing K-S Distance.

$$K-S(MC_j) = \max \{ [CDF(M_{ij}/(F_i \leq FC))] - [CDF(M_{ij}/(F_i > FC))] \} \quad (2)$$

The history of changes (e.g., requirements, design, and code) and other activities (e.g., inspections, tests, and failure and fault observations) are recorded at the beginning of a module's listing (i.e., *prologue*). The number of lines in this section is called the *prologue size*. Because this metric records the volatility of the software, it is a very good quality discriminator, as our results will demonstrate. A *statement* is an executable statement in the Hal/S programming language that is used to code the *Space Shuttle* flight software.

Table 2. Kolmogorov-Smirnov Distance for $drcount=0$ vs. $drcount>0$ Validation Sample 1 (n=100 modules)					
Metric (symbol)	Definition (counts per module)	Critical Value	Distance	α	Rank
prologue size (P)	change history line count in module listing	38	0.585	0.005	1
statements (S)	executable statement count	26	0.557	0.005	2
eta1 (E1)	unique operator count	10	0.492	0.005	3
nodes (N)	node count (in control graph)	11	0.487	0.005	4

3.2. Stage 2: perform contingency table analysis

3.2.1. Validation contingency table

For each BDF identified in Stage 1 we use the *Contingency Table* (see Table 3) and its accompanying χ^2 statistic [Conover 1971] to further evaluate the ability of the functions to discriminate high quality from low quality, from both statistical (e.g., values of χ^2 and α) and application (e.g., ability of the metric set to correctly classify low quality modules) standpoints. In

Table 3, MC_j and FC classify modules into one of four categories. The left column contains modules where none of the metrics exceeds its critical value; this condition is expressed with a Boolean AND function of the metrics. This is the *ACCEPT* column, meaning that according to the classification decision made by the metrics, these modules have acceptable quality. The right column contains modules where at least one metric exceeds its critical value; this condition is expressed by a Boolean *OR* function of the metrics. This is the *REJECT* column, meaning that according to the classification decision made by the metrics, these modules have unacceptable quality. The top row contains modules that are high quality; these modules have a quality factor that does not exceed its critical value (e.g., $drcount=0$). The bottom row contains modules that are low quality; these modules have a quality factor that exceeds its critical value (e.g., $drcount>0$).

Equation (3) gives the algorithms for making the cell counts of modules, using the BDFs of F_i and M_{ij} that are computed over the n modules for m metrics. This equation is an implementation of the relation given in (1).

$$\begin{aligned}
C_{11} &= \text{COUNT}_{i=1}^n \text{ FOR } ((F_i \leq FC) \wedge (M_{i1} \leq MC_1) \dots \wedge (M_{ij} \leq MC_j) \dots \wedge (M_{im} \leq MC_m)) \\
C_{12} &= \text{COUNT}_{i=1}^n \text{ FOR } ((F_i \leq FC) \wedge ((M_{i1} > MC_1) \dots \vee (M_{ij} > MC_j) \dots \vee (M_{im} > MC_m))) \\
C_{21} &= \text{COUNT}_{i=1}^n \text{ FOR } ((F_i > FC) \wedge (M_{i1} \leq MC_1) \dots \wedge (M_{ij} \leq MC_j) \dots \wedge (M_{im} \leq MC_m)) \\
C_{22} &= \text{COUNT}_{i=1}^n \text{ FOR } ((F_i > FC) \wedge ((M_{i1} > MC_1) \dots \vee (M_{ij} > MC_j) \dots \vee (M_{im} > MC_m)))
\end{aligned} \tag{3}$$

for $j=1, \dots, m$, and where $\text{COUNT}(i)=\text{COUNT}(i-1)+1$ FOR Boolean expression *true* and $\text{COUNT}(i)=\text{COUNT}(i-1)$, otherwise; $\text{COUNT}(0)=0$.

The counts correspond to the cells of the *Contingency Table* (C_{11} , C_{12} , C_{21} , and C_{22}), as shown in Table 3, where row and column totals are also shown: n , n_1 , n_2 , N_1 , and N_2 . The analysis could be generalized to include multiple quality factors, if necessary; in this case, the *Contingency Table* would have more than two rows.

In addition to counting modules in Table 3, we must also count the quality factor (e.g., $drcount$) that is incorrectly classified. This is shown as Remaining Factor, RF, in the *ACCEPT* column. This is the quality factor count on modules that should have been rejected. Also shown is Total Factor, TF, the total quality factor count on all the modules in the sample (i.e., the sum of $drcount$). Lastly we show RFM (Remaining Factor Modules) that is the count of modules with quality factor count >0 (i.e., modules with Remaining Factor, RF).

Table 3 and subsequent equations show an example validation, where the optimal combination of

metrics from Table 2 and their critical values for a random sample of 100 modules (sample 1), from the population of 1397, is *prologue size* (P) with a critical value of 38 and *statements* (S) with a critical value of 26. This low value of *statements* is understandable because the median value in the builds analyzed is 23. There are many small modules that call a subroutines, compute a value, and transfer control to another module. Later we will explain how we arrived at this particular combination of metrics as the optimal set.

3.2.2. Statistical criteria

We validate a BDF statistically by demonstrating that it partitions Table 3 in such a way that C_{11} and C_{22} are large relative to C_{12} and C_{21} . If this is the case, a large number of high quality modules (e.g., modules with $drcount=0$) would have $M_{ij} \leq MC_j$ and would be correctly classified as high quality. Similarly, a large number of low quality modules (e.g., modules with $drcount>0$) would have $M_{ij} > MC_j$ and would be correctly classified as low quality. One measure of the degree to which this is the case is estimated by the chi-square (χ^2) statistic [Conover 1971]. If computed $\chi^2_c > \chi^2_s$ (chi-square at specified α_s) and if computed $\alpha_c < \alpha_s$, then these results suggest that a given BDF can discriminate between high and low quality. However, because the χ^2 test may not produce consistent results [Eman 1998], we use it only as *one* of several indicators of *Discriminative Power*. Other criteria are misclassification rates and, most important, application criteria (see below). We note that the use of chi-square and alpha as statistical criteria is independent of the application (i.e., these criteria could be used whether the application is metrics or personnel management). Application criteria, on the other hand, such as *Quality* and *Inspection* (see below) are meaningful in the *context* of the metrics application.

3.2.2.1. Misclassification

We compute the degree of misclassification in Table 3 by noting that ideally $C_{11}=n_1=N_1$, $C_{12}=0$, $C_{21}=0$, $C_{22}=n_2=N_2$. The extent that this is not the case is estimated by *Type 1* misclassifications (i.e., the module has *Low Quality* and the metrics "say" it has *High Quality*) and *Type 2* misclassifications (i.e., the module has *High Quality* and the metrics "say" it has *Low Quality*). Thus, we define the following measures of misclassification:

$$\begin{aligned} \text{Proportion of modules of Type 1: } P_1 &= C_{21}/n & (4) \\ \text{For the example, } P_1 &= (1/100)*100=1\%. \end{aligned}$$

$$\begin{aligned} \text{Proportion of modules of Type 2: } P_2 &= C_{12}/n & (5) \\ P_2 &= (27/100)*100=27\%. \end{aligned}$$

$$\begin{aligned} \text{Proportion of modules of Type 1+Type 2: } P_{12} &= (C_{21}+C_{12})/n & (6) \\ P_{12} &= ((1+27)/100)*100=28\%. \end{aligned}$$

Table 3. Validation Contingency Table			
	$\wedge(M_{ij} \leq MC_j)$ $P_i \leq 38 \wedge S_i \leq 26$	$\vee(M_{ij} > MC_j)$ $P_i > 38 \vee S_i > 26$	
High Quality $F_i \leq FC$ $drcount=0$	$C_{11}=30$	$C_{12}=27$ Type 2	$n_1=57$
Low Quality $F_i > FC$ $drcount>0$	$C_{21}=1$ Type 1	$C_{22}=42$	$n_2=43$
	$N_1=31$ RF=1, RFM=1	$N_2=69$	$n=100$ TF=192
	ACCEPT	REJECT	

3.2.3. Application criteria

It is insufficient to validate only with respect to statistical criteria. In the final analysis, it is the performance of the metrics in the application context that counts. Therefore, we validate metrics with respect to the application criteria: *Quality and Inspection*, which are related to quality achieved and the cost to achieve it, respectively [Schneidewind 1997a; Schneidewind 1997b]. At the Design phase of Build 2 in Figure 1, we predict that the quality computed by equations (7)--(12) will be delivered to maintenance, assuming that the modules that are rejected by the quality control process are inspected and tested and that the problems that are found are corrected. Furthermore, we predict that the degree of inspection computed by equation (13) will be required to achieve this quality.

3.2.3.1. Quality

First, we estimate the ability of the metrics to correctly classify quality, given that the quality is known to be low:

$$\text{LQC: Proportion of low quality (e.g., } drcount>0) \text{ modules correctly classified} = C_{22}/n_2 \quad (7)$$

For the example, $\text{LQC} = (42/43) * 100 = 97.7\%$.

Second, we estimate the ability of the metrics to correctly classify quality, given that the BDF has classified modules as *ACCEPT*. This is done by summing quality factor in the *ACCEPT* column in

Table 3 to produce Remaining Factor, RF (e.g., remaining *drcount*), given by equation (8).

$$RF = \sum_{i=1}^n F_i \text{ FOR } ((F_i > FC) \wedge (M_{i1} \leq MC_1) \dots \wedge (M_{ij} \leq MC_j) \dots \wedge (M_{im} \leq MC_m)), \text{ for } j=1, \dots, m \quad (8)$$

This is the sum of quality factor F_i (e.g., *drcount*) on modules incorrectly classified as high quality because $(F_i > FC) \wedge (M_{ij} \leq MC_j)$ for these modules. We assume that the elements of F_i are additive and that the lower its value, the higher the quality of the module. This would be the case for any quality factor of interest in this analysis: discrepancy report count, error count, fault count, and failure count.

We estimate the proportion of RF by equation (9), where TF is the total quality factor F_i for the Validation Sample.

$$RFP = RF/TF \quad (9)$$

For the example, from Table 3 there is a one DR on one module that is incorrectly classified (i.e., $RF=1$). The total number of DRs for the 100 modules is 192. Therefore, $RFP=(1/192)*100=.52\%$.

We estimate the density of RF by equation (10).

$$RFD = RF/n \quad (10)$$

For the example, $RFD=1/100=.01$ *drcount/module*.

In addition, we estimate the count of modules that were incorrectly classified because they have DRs written against them (i.e., have $F_i > FC$). The proportion remaining RMP is given by equation (11). Note that $RMP=P_1$ (proportion of *Type 1* misclassifications) when $FC=0$ (i.e., the only modules with $F_i > 0$ will be in the C_{21} cell); see Table 3.

$$RMP = RFM/n, \quad (11)$$

where RFM is given by:

$$RFM = \sum_{i=1}^n \text{COUNT FOR } ((F_i > 0) \wedge (M_{i1} \leq MC_1) \dots \wedge (M_{ij} \leq MC_j) \dots \wedge (M_{im} \leq MC_m)), \text{ for } j=1, \dots, m. \quad (12)$$

For the example, there is one accepted module with one DR, so $RMP=(1/100)*100=1\%$.

3.2.3.2. Inspection

Inspection is one of the costs of high quality. We are interested in weighing inspection requirements (i.e., percent of modules rejected and subjected to detailed inspection) against the quality that is achieved, for various BDFs. We estimate inspection requirements by noting that all modules in the *REJECT* column of Table 3 must be inspected; this is the count $C_{12}+C_{22}$. Thus, the proportion of modules that must be inspected is given by:

$$I=(C_{12}+C_{22})/n \quad (13)$$

For the example, $I=((27+42)/100)*100=69\%$ and the percentage accepted is $1-I = 31\%$.

3.2.4. Summary of validation results

The results of the validation example are summarized in Table 4. The properties of *dominance* and *concordance* are evident in these validation results and in other samples we have analyzed from this data. That is, a point is reached in adding metrics where *Discriminative Power* is not increased because: 1) the contribution of the dominant metrics in correctly classifying quality has already taken effect and 2) additional metrics essentially replicate the classification results of the dominant metrics -- the *concordance* effect. This result is due to the property of the BDF used as an OR function, which will cause a module to be rejected if only one of the module's metrics exceeds its critical value. These effects can only be observed if a marginal analysis is performed, where metrics are added to the set one-by-one and the calculations shown in Table 4 are made after each metric is added. For each added metric, its effect is evaluated with respect to both statistical and application criteria. In addition, a suitable stopping rule must be used to know when to stop adding metrics (see the next section).

Table 4. *Discriminative Power* Validity Evaluation (Sample 1, n=100 modules)

Metric Set	Critical Values				Statistical Criteria				Application Criteria			
	P	S	E1	N	P ₁ %	P ₂ %	χ^2_c	α_c for χ^2_c	LQC %	RFP %	RMP %	I %
P	38				2	21	33.2	8.4×10^{-9}	95.3	1.56	2	62
P,S	38	26			1	27	26.7	2.4×10^{-7}	97.7	0.52	1	69
P,S,E1	38	26	10		1	30	22.5	2.1×10^{-6}	97.7	0.52	1	72
K-S Distance	0.585	0.557	0.492	0.487								

P: prologue size, S: statements, E1: etal, N: nodes

3.3. Stage 3: Apply a stopping rule for adding metrics

One rule for stopping the addition of metrics to a BDF is to quit when RFP no longer decreases as metrics are added. This is the *maximum quality* rule. This rule is illustrated in Table 4. When a third metric, *etal*(E1), is added, there is no decrease in RFP and RMP nor is there an increase in LQC. If it is important to strike a balance between quality and cost (i.e., between RFP and I), we add metrics until the ratio of the relative change in RFP to the relative change in I is maximum, as given by the *Quality Inspection Ratio* (QIR) in equation (14), where *i* refers to the previous RFP and I:

$$QIR = (\Delta RFP / RFP_i) / (\Delta I / I_i) \quad (14)$$

For the example, $QIR(P \rightarrow P,S) = ((|.52 - 1.56|) / 1.56) / ((69 - 62) / 62) = 5.90$. This is the value of QIR in going from one metric *prologue size* (P) to two metrics (P,S), adding *statements* (S).

Also, $QIR(P,S \rightarrow P,S,E1) = 0$. This is the value of QIR in going from two metrics (P,S) to three metrics (P,S, E1), adding *etal* (E1).

Therefore, we stop adding metrics after *statements* has been added. In this particular case, equation (13) produces the same metric set as the *maximum quality* rule.

4. Comparison of validation with application results

In order to compare validation with application results, we first show how the Contingency Table looks at the Design phase of Build 2 in Figure 1, when only the metrics M_{ij} and their critical values MC_j are available. This is shown in Table 5, where the "?" indicates that the quality factor data F_i are not available when the validated metrics are used in the quality control function of Build 2. During the Design phase of Build 2, modules are classified according to the criteria that have been described. A second disjoint random sample of 100 modules (sample 2) was used to illustrate the process. Whereas 31 and 69 modules were accepted and rejected, respectively, during Build 1, 40 and 60 modules were accepted and rejected, respectively, during Build 2. The rejected modules

would be given priority attention (i.e., subjected to rigorous inspection).

Table 5. Application Contingency Table			
	$\wedge(M_{ij} \leq MC_j)$ $P_i \leq 38 \wedge S_i \leq 26$	$\vee(M_{ij} > MC_j)$ $P_i > 38 \vee S_i > 26$	
High Quality ?	?	Type 2 ?	?
Low Quality ?	Type 1 ?	?	?
	$N_1=40$	$N_2=60$	$n=100$
	ACCEPT	REJECT	

A comparison of the Validation Sample (Build 1) with the Application Samples (Build 2) with respect to statistical criteria is shown in Table 6. A comparison of the Validation Sample with the Application Samples with respect to application criteria is shown in Tables 7 and 8. As we have mentioned, only metrics data is available when the validated metrics are applied during the Design phase of Build 2 in Figure 1. However, to have a basis for comparison with the validation results, we computed the values shown in Tables 6, 7, and 8 *retrospectively* (i.e., after Build 2 was far enough along to be able to collect all of the quality factor data at the conclusion of the Test phase). The values for samples 2, 3, and 4 in Tables 7 and 8 are the *actual* quality delivered to maintenance, as shown during the Test phase of Figure 1. The reader should compare the results of Samples 2, 3, and 4 with those of Sample 1 in the tables. As the accuracy of classification of low quality software increases, the accuracy of classifying high quality software decreases and inspection cost increases. However, the more important consideration is to prevent low quality software from being delivered to maintenance, particularly in safety critical systems like the *Space Shuttle*.

Table 6. Statistical Criteria P1 and P2 for Metric Set: P,S Validation (Sample 1) vs. Application (Samples 2, 3, and 4), n=100 modules							
P1 : Percentage Type 1 Misclassification				P2: Percentage Type 2 Misclassification			
Sample 1	Sample 2	Sample 3	Sample 4	Sample 1	Sample 2	Sample 3	Sample 4
1.0	1.0	4.0	3.0	27.0	24.0	18.0	22.0

Table 7. Application Criteria LQC and RFP for Metric Set: P,S Validation (Sample 1) vs. Application (Samples 2, 3, and 4), n=100 modules							
LQC: Percentage of low quality modules (<i>drcount</i> >0) correctly classified				RFP: Percentage of quality factor (<i>drcount</i>) incorrectly classified			
Sample 1	Sample 2	Sample 3	Sample 4	Sample 1	Sample 2	Sample 3	Sample 4
97.7	97.3	91.1	93.2	0.52	.62	3.01	1.50

Table 8. Application Criteria RFD and I for Metric Set: P,S Validation (Sample 1) vs. Application (Samples 2, 3, and 4), n=100 modules							
RFD: Density of quality factor (<i>drcount</i> /module) incorrectly classified				I: Percentage of modules inspected			
Sample 1	Sample 2	Sample 3	Sample 4	Sample 1	Sample 2	Sample 3	Sample 4
.01	.01	.05	.03	69	60	59	63

5. Quality point and confidence interval estimates

In addition to the quantities in Tables 3 -- 8, there are other quantities of interest, such as proportion of modules with zero and non-zero *drcount* and their confidence intervals. For these quantities, software developers and maintainers are provided with both point estimates and interval estimates of the range in which the actual quality values are likely to fall. Thus, they are able to anticipate rather than react to quality problems. For example, estimates obtained from Build 1 in Figure 1 are used to predict the quality of software that would be delivered to maintenance if corrective action were not taken. This action is the quality control step of the Design Phase of Build 2 where modules are rejected and subjected to detailed inspection and test if their metrics values exceed the critical values. In addition, the estimates provide indications of resource levels that are needed to achieve quality goals. For example, if the predicted quality of the software were lower than the specified quality, the difference would be an indication of increased usage of personnel and computer time during inspection and testing, respectively.

A benefit of using confidence limits is that they provide protection against prediction error. A prediction error could arise because the very act of measuring and predicting may affect the predictions -- the *Heisenberg Principle*. For example, *prologue size*, the record of change history, has proven to be a good predictor of quality. However, if the software is changed in response to problems observed during the quality control function, thereby adding to the change history and *prologue size*, this effect would tend to make the original predictions optimistic. Another protection against prediction error is to periodically repeat the predictions as the software evolves over the life cycle.

The normal approximation to the binomial distribution is used to estimate the confidence limits of the proportions. This distribution is used because we are interested in estimating the proportions of modules and *drcount* that fall into one of two categories (i.e., a module is either accepted or rejected or DRs are either present or not present on a module). The normal approximation gives the mean proportion p of modules or DRs that fall into one of two categories and the confidence limits are a function of p .

The point and confidence limit estimates for module and quality factor counts use terms that are defined below. Where it is necessary to distinguish validation from application quantities in the computations, we use primed notation for the latter.

n : number of modules in the Validation and Application samples (see Tables 3 and 5, respectively)

N_1 : number of modules accepted in the Validation Sample of Build 1

N_2 : number of modules rejected in the Validation Sample of Build 1

N_1' : number of modules accepted in the Application Samples of Build 2

N_2' : number of modules rejected in the Application Samples of Build 2

5.1. Module counts

Module count estimates are made using the Validation Sample in the Test Phase of Build 1. These estimates are applied to the Application Samples in the Design Phase of Build 2 and compared with actual values in Table 9.

The proportion of *all* modules with quality factor $F_i > 0$ (e.g., *drcount* > 0 on module i) in the *entire* Validation Sample is given by equation (15):

$$p_n = (\text{COUNT}_{i=1}^n \text{ FOR } F_i > 0) / n \quad (15)$$

where $\text{COUNT}(i) = \text{COUNT}(i-1) + 1$ FOR expression *true* and $\text{COUNT}(i) = \text{COUNT}(i-1)$, otherwise; $\text{COUNT}(0) = 0$. We use this equation to estimate p_n' in the Application samples. We obtain the two-sided confidence interval of p_n from expression (16). We use this expression to estimate the lower and upper limits of p_n' in the Application Samples:

$$p_n \pm Z_{\alpha/2} \sqrt{\frac{(p_n)(1-p_n)}{n}} \quad (16)$$

As shown in Table 9, we would expect the proportion of *all* modules with *drcount* > 0 in maintenance to be between 33.3%-52.7% unless corrective action is taken to make these limits lower. If corrective action is taken, this estimate provides bounds on the resources -- personnel and computer time -- that would be required to inspect, correct, and test defective modules.

The proportion of *accepted* modules with quality factor $F_i > 0$ (e.g., $drcount > 0$ on module i) in the Validation Sample is given by equation (17), where RFM is obtained from equation (12):

$$pN_1 = \text{RFM} / N_1 \quad (17)$$

We use this equation to estimate pN_1' in the Application samples. We obtain the one-sided upper confidence limit of pN_1 from expression (18). We use this expression to estimate the upper limit of pN_1' in the Application Samples:

$$pN_1 + Z_{\alpha} \sqrt{\frac{(pN_1)(1 - pN_1)}{N_1}} \quad (18)$$

As shown in Table 9, we would expect the proportion of *accepted* modules with $drcount > 0$ in maintenance to be $\leq 8.45\%$ as the result of the quality control effort in the Design Phase of Build 2.

The proportion of *rejected* modules with quality factor $F_i > 0$ (e.g., $drcount > 0$ on module i) in the Validation Sample is given by equation (19):

$$pN_2 = ((p_n)(n) - (\text{RFM})) / N_2 \quad (19)$$

This is equal to: (*all* modules with quality factor $F_i > 0$) minus (*accepted* modules with quality factor $F_i > 0$), divided by the number of rejected modules. We use this equation to estimate pN_2' in the Application Samples. We obtain the one-sided lower confidence limit of pN_2 from expression (20). We use this expression to estimate the lower limit of pN_2' in the Application Samples:

$$pN_2 - Z_{\alpha} \sqrt{\frac{(pN_2)(1 - pN_2)}{N_2}} \quad (20)$$

As shown in Table 9, we would expect the proportion of *rejected* modules with $drcount > 0$ in maintenance to be $\geq 51.2\%$ as the result of the quality control effort in the Design Phase of Build 2.

5.2. Quality factor counts

Quality factor *proportion* count estimates in (21), ..., (24) are made using the Validation Sample in the Test Phase of Build 1. Quality factor *total* count estimates in (25) and (26) use data from the Validation Sample and data that is available in the Application Samples in the Design Phase of Build 2: number of modules *accepted*, N_1' and number of modules *rejected*, N_2' . These estimates are applied to the Application Samples in the Design Phase of Build 2 and compared with actual values in Tables 9 and 10.

The proportion of quality factor $F_i > 0$ (e.g., $drcount > 0$) that occurs on *accepted* modules in the Validation Sample is given by equation (21):

$$d_1 = RF/TF \quad (21)$$

where RF is obtained from equation (8) and TF is the total quality factor F_i for the Validation Sample. We use this equation to estimate d_1' in the Application samples. We obtain the one-sided upper confidence limit of d_1 from expression (22). We use this expression to estimate the upper limit of d_1' in the Application Samples:

$$d_1 + Z_\alpha \sqrt{\frac{(d_1)(1-d_1)}{TF}} \quad (22)$$

As shown in Table 9, we would expect the proportion of $drcount > 0$ on *accepted* modules in maintenance to be $\leq 1.38\%$ as the result of the quality control effort in the Design Phase of Build 2.

The proportion of quality factor $F_i > 0$ (e.g., $drcount > 0$) that occurs on *rejected* modules in the Validation Sample is given by equation (23):

$$d_2 = 1 - d_1 \quad (23)$$

We use this equation to estimate d_2' in the Application Samples. We obtain the one-sided lower confidence limit of d_2 from expression (24). We use this expression to estimate the lower limit of d_2' in the Application Samples:

$$d_2 - Z_\alpha \sqrt{\frac{(d_2)(1-d_2)}{TF}} \quad (24)$$

As shown in Table 9, we would expect the proportion of $drcount > 0$ on *rejected* modules in maintenance to be $\geq 98.6\%$ as the result of the quality control effort in the Design Phase of Build 2.

The total quality factor $F_i > 0$ (e.g., $drcount > 0$) that occurs on *accepted* modules in the Validation Sample is given by equation (25):

$$D_1 = (RF/N_1)(N_1') \quad (25)$$

We use this equation as a predictor of D_1' in the Application Samples. As shown in Table 10, we would expect the *total drcount* on *accepted* modules in maintenance to be 1.29, 1.32, and 1.19 for Application Samples 2, 3, and 4, respectively. The reason for the three estimates of Sample 1 is that each sample has a different number of *accepted* modules N_1' in equation (25).

The total quality factor of $F_i > 0$ (e.g., $drcount > 0$) that occurs on *rejected* modules in the Validation Sample is given by equation (26):

$$D_2 = ((TF - RF)/N_2)(N_2') \quad (26)$$

We use this equation as a predictor of D_2' in the Application Samples. As shown in Table 10, we would expect the *total drcount* on *rejected* modules in maintenance to be 166.1, 163.3, and 174.4 for

Application Samples 2, 3, and 4, respectively. The reason for the three estimates of Sample 1 is that each sample has a different number of *rejected* modules N_2' in equation (26).

Ten of the actual values out of the fifteen cases in Table 9 fall within the confidence limits. The average relative error across six comparisons between Sample 1 versus Samples 2, 3, 4 in Table 10 is 28.9% with a standard deviation of 30.7%. Variation in results *may* be caused by sampling error (i.e., in order to obtain disjoint samples, it was necessary to sample without replacement).

Table 9. Validation Predictions (Sample 1) vs. Application Actual Values (Samples 2, 3, and 4)					
	Point Estimates (Sample 1)	95% Confidence Limits (Sample 1)	Actual Values		
			Sample 2	Sample 3	Sample 4
p_n' : proportion of <i>all</i> modules with <i>drcount</i> >0	43.0%	33.3%-52.7%	37.0%	45.0%	44.0%
pN_1' : proportion of <i>accepted</i> modules with <i>drcount</i> >0	3.22%	LE 8.45%	2.50%	9.76%	8.11%
pN_2' : proportion of <i>rejected</i> modules with <i>drcount</i> >0	60.9%	GE 51.2%	60.0%	69.5%	65.1%
d_1' : proportion of <i>drcount</i> >0 on <i>accepted</i> modules	.52%	LE 1.38%	.62%	3.01%	1.50%
d_2' : proportion of <i>drcount</i> >0 on <i>rejected</i> modules	99.5%	GE 98.6%	99.4%	97.0%	98.5%

Table 10. Validation Actual Values and Predictions (Sample 1) vs. Application Actual Values (Samples 2, 3, and 4)							
	Actual Sample 1	Estimate Sample 1	Actual Sample 2	Estimate Sample 1	Actual Sample 3	Estimate Sample 1	Actual Sample 4
D_1' : <i>total drcount</i> on <i>accepted</i> modules	1	1.29	1	1.32	5	1.19	3
D_2' : <i>total drcount</i> on <i>rejected</i> modules	191	166.1	160	163.3	161	174.4	197

6. Comparison of Boolean and linear discriminant functions

We compared the quality classifying ability during validation of the Boolean discriminant function (BDF) with an alternate method: the linear discriminant function (LDF) consisting of the summation across metrics of the product of standardized metrics variables and standardized classification coefficients [Jobson 1992]. For the BDF, we used the optimal metrics set -- *prologue size* and *statements* -- and results obtained from Table 4. For the LDF, we used the set of nine

metrics listed in Table 11 and a marginal analysis that yielded the highest *Discriminative Power* as measured by the Eigenvalue and χ^2 . The comparison is shown in Table 11. In the comparison, we used both statistical and application criteria. In the application category, we did not compute RFP and RMP for the LDF as we did in Table 4. Unlike the BDF where equations (8) and (9) count quality factor and (11) and (12) count modules that are misclassified into the *ACCEPT* category, there is no algorithm for making these computations for the LDF. It would have been necessary to compare the metrics and *drcount* for each module with the LDF to determine how the metrics classified the modules and *drcount*. However, a good comparison is obtained by using LQC. In this example, Table 10 shows that the BDF does a better job of classifying the low quality modules (e.g., lower value of P_1 and higher value of LQC) and that LDF does a better job of classifying the high quality modules (e.g., lower values of P_2 and I). As stated in Section 1, the reason for this result is that BDFs make fewer mistakes in classifying software that is low quality than is the case when linear vectors of metrics are used because the critical values provide additional information for discriminating quality. The implications for applying the validated metrics during the quality control function of the Design Phase of Build 2 is that the BDF would yield higher quality and the LDF would yield lower cost. Our preference is the BDF in a safety critical system like the *Space Shuttle*, where high quality software is the paramount objective.

Table 11. Comparison of Boolean Discriminant Function (BDF) with Linear Discriminant Function (LDF) Validity Evaluation (Sample 1, n=100 modules)							
		Statistical Criteria				Application Criteria	
Function	Metric Set	P_1 %	P_2 %	χ^2_c	α_c for χ^2_c	LQC %	I %
BDF	P,S	1.0	27.0	26.7	2.4×10^{-7}	97.7	69.0
LDF	9 Metrics	9.0	9.0	37.5	≈ 0	79.1	43.0

LDF Metric Set (counts per module): Halstead eta1, eta2, η_1 , and η_2 ; lines of code, prologue size, nodes, paths, and maximum path.

7. Metric characteristics of failed modules

Further evidence of the model's ability to identify low quality during development is shown in Table 12. This table shows the 15 modules that failed during maintenance of the 1397 modules of Build 2 in Figure 1, where the severity of the 10 failures decreases from 2 to 4. In the case of failure # 7, six modules caused this failure. The table also shows the module metrics and validated critical values that were obtained during Build 1. For all failed modules, one or more of their metric values

exceed the critical value. Metric values in *italics* would fail to reject these modules during quality control of the Design phase of Build 2. However, this would be compensated for by the metric *prologue size* that would have correctly rejected all of these modules. To illustrate the difference in metric characteristics of the failed modules versus all the modules of Build 2, the means of each were computed. The difference in means is significant at $\alpha < .05$. As this example illustrates, although a metrics program can alert the developer to the possibility of unreliable software, it cannot *prevent* failures from occurring. In this example, the inspection and test process failed to find and correct the problems before Build 2 entered maintenance.

Table 12. Metric Characteristics of Failed Modules

Failure Number	Severity Level	Module ID	prologue size	statements	eta1	nodes	drcount
1	2	13	493	738	46	394	22
2	3	974	299	192	31	98	2
3	2	1286	115	110	28	48	5
4	3	711	205	1	5	96	6
5	3	1300	82	3	8	20	1
6	3	515	851	875	44	529	15
7	2	464	69	15	16	12	4
7	2	465	76	30	24	21	4
7	2	466	68	15	16	12	4
7	2	467	72	30	24	21	2
7	2	468	153	10	11	75	3
7	2	472	100	1	6	40	1
8	4	555	943	819	34	174	26
9	3	904	122	128	31	64	1
10	4	882	157	107	30	51	5
Critical Value			38	26	10	11	0
Failed Modules Mean			253.7	204.9	23.6	110.3	6.7
Build 2 Mean			134.6	70.2	16.7	28.4	1.8

8. Conclusions

A model was developed for controlling and predicting the quality of software that is delivered by development to maintenance. The model provides software developers and maintainers with both point estimates and interval estimates of the range in which the actual quality values are likely to fall. Thus, they are alerted to the need to take corrective action.

It is important when validating and applying metrics to consider both statistical and application criteria and to measure the marginal contribution of each metric in satisfying these criteria. When this approach is used, we observe that a point is reached where adding metrics makes no contribution to improving quality and the cost of using additional metrics increases. This phenomenon is due to the metric classification properties of *dominance* and *concordance*. Using our approach, we achieved an error of $\leq 3\%$ in classifying quality factors for the samples used in the study. The ratio of the relative improvement in quality to the relative increase in inspection cost is a new and effective stopping rule for adding metrics.

Our Boolean discriminant function (BDF) is a new type of discriminant for classifying software quality to support an *integrated* approach to control and prediction in one model, and our application of Kolmogorov-Smirnov distance is a new way to determine a metric's critical value. On this application, the BDF, using two metrics, was superior to a linear discriminant function, using nine metrics, in classifying low quality software; however, when used for quality control, the BDF requires more inspection.

Finally, with a very limited sample of modules that caused failures we found that the validated metrics, if they had been applied to the modules that eventually failed, would have acted as early indicators of these failures.

Acknowledgments

We wish to acknowledge the support provided for this project by Dr. William Farr of the Naval Surface Warfare Center and Dr. Allen Nikora of the Jet Propulsion Laboratory; the data provided by Prof. John Munson of the University of Idaho; the data and assistance provided by Ms. Julie Barnard of United Space Alliance; the helpful comments of Dr. Linda Rosenberg of NASA, Goddard; and the anonymous reviewers.

References

Briand, Lionel C., John Daly, Victor Porter, and Jürgen Wüst (1998), "Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems", *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 334-343.

Conover, W. J. (1971), *Practical Nonparametric Statistics*, John Wiley & Sons, Inc., New York, N.Y.

Eman, Khaled El (1998), "The Predictive Validity Criterion for Evaluating Binary Classifiers",

Proceedings of the Fifth International Metrics Symposium, IEEE Computer Society Press, Los Alamitos, CA, pp. 235-244.

IEEE Standard for a Software Quality Metrics Methodology (1998), Revision, IEEE Std 1061, IEEE Standards Office, Piscataway, NJ.

Jobson, J. D. (1992), *Applied Multivariate Data Analysis*, Volume II, Springer-Verlag, New York, NY.

Khoshgoftaar, Taghi, M. and Edward B. Allen (1998), "Predicting the Order of Fault-Prone Modules in Legacy Software", *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 344-353.

Khoshgoftaar, Taghi, M. and Edward B. Allen (1997), "Logistic Regression Modeling of Software Quality", TR-CSE-97-24, Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, FL.

Khoshgoftaar, Taghi, M. , Edward B. Allen, Robert Halstead, and Gary P. Trio (1996), "Detection of Fault-Prone Software Modules During a Spiral Life Cycle", *Proceedings of the International Conference on Software Maintenance*, IEEE Computer Society Press, Los Alamitos, CA, pp. 69-76.

Khoshgoftaar, Taghi, M. , Edward B. Allen, Kalai Kalaichelvan, and Nishith Goel (1996), "Early Quality Prediction: A case Study in Telecommunications", *IEEE Software*, 13, 1, 65-71.

Nikora, Allen P. and John C. Munson (1998), "Determining Fault Insertion Rates for Evolving Software Systems", *Proceedings of the Ninth International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, pp. 306-315.

Ohlsson, Niclas and Hans Alberg (1996), "Predicting Fault-Prone Software Modules in Telephone Switches", *IEEE Transactions on Software Engineering*, 22, 12, 886-894.

Porter, A. A. and R. W. Selby (1990), "Empirically Guided Software Development Using Metric-Based Classification Trees", *IEEE Software*, 7, 2, 46-54.

Schneidewind, Norman F. (1997a), "Software Metrics Model for Quality Control", *Proceedings of the International Metrics Symposium*, IEEE Computer Society Press, Los Alamitos, CA, pp. 127-136.

Schneidewind, Norman F., (1997b) "Software Metrics Model for Integrating Quality Control and Prediction", *Proceedings of the International Symposium on Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, pp. 402-415.

Schneidewind, Norman F., (1995), "Work in Progress Report: Experiment in Including Metrics in a Software Reliability Model", *Proceedings of the Annual Oregon Workshop on Software Metrics*, Portland State University, Portland, OR, 17 pages.

Schneidewind, Norman F., (1992), "Methodology for Validating Software Metrics", *IEEE Transactions on Software Engineering*, 18, 5, 410-422.

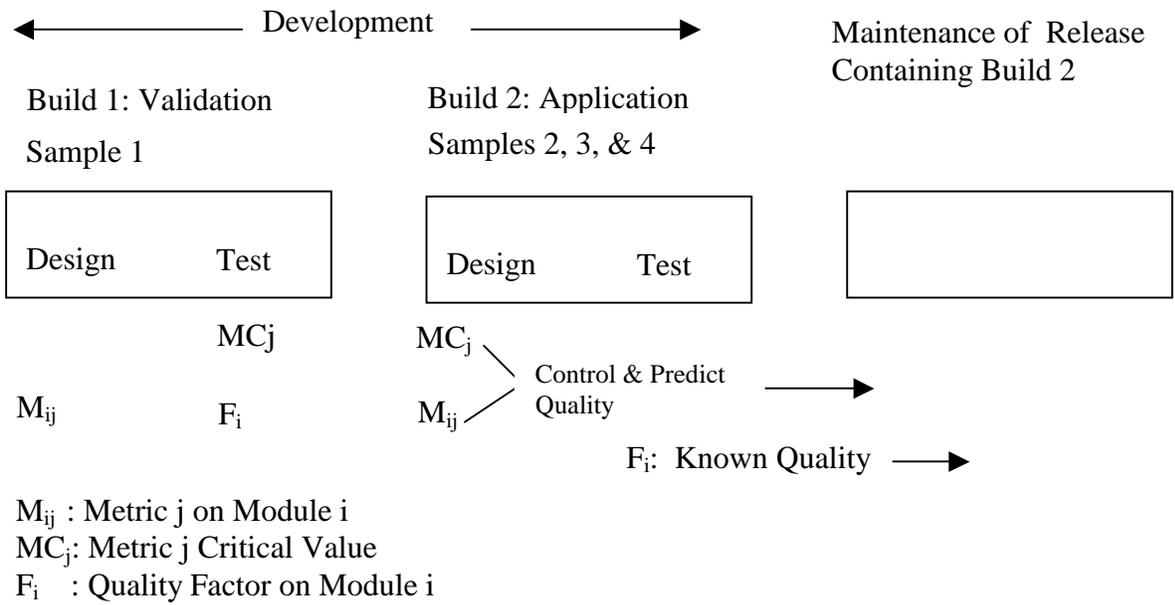


Figure 1. Measurement Process

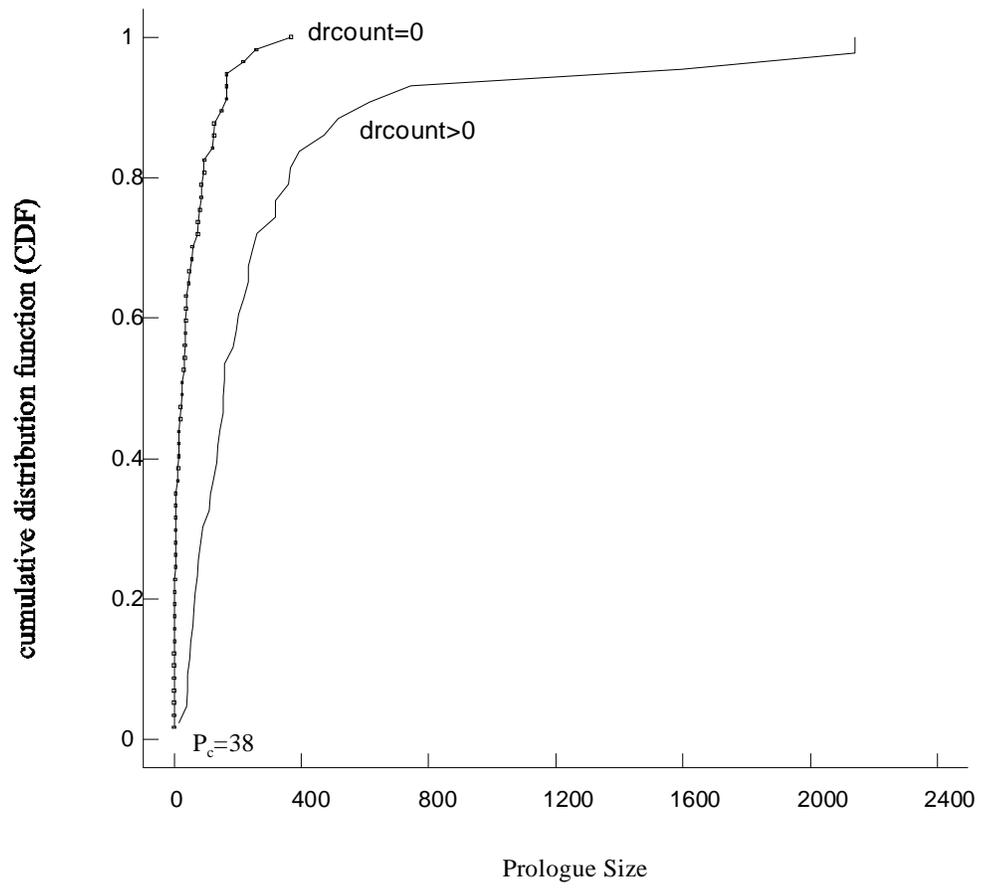


Figure 2. K-S Test: Prologue Size CDF (sample 1, n=100 modules)