

# Investigation of Logistic Regression as a Discriminant of Software Quality

Proceedings of the 7th International Software Metrics Symposium, 4-6 April 2001,  
London, England, pp 328-337.

Norman F. Schneidewind  
Naval Postgraduate School  
email: [nschneid@nps.navy.mil](mailto:nschneid@nps.navy.mil)

## Abstract

*We investigated the possibility that Logistic Regression Functions (LRFs), when used in combination with Boolean Discriminant Functions (BDFs), which we had previously developed, would improve the quality classification ability of BDFs when used alone. This was the case; when the union of a BDF and LRF was used to classify quality, the predicative accuracy of quality and inspection cost was improved over that of using either function alone for the Space Shuttle. Also, the LRFs proved useful for ranking the quality of modules in a build. The significance of these results is that very high quality classification accuracy (1.25% error) can be obtained while reducing the inspection cost incurred in achieving high quality. This is particularly important for safety critical systems. Because the methods are general and not particular to the Shuttle, they could be applied to other domains. A key part of the LRF development was a method for identifying the critical value (i.e. threshold) that could discriminate between high and low quality and at the same time constrain the cost of inspection to a reasonable value.*

**Key Words:** Software quality prediction, Logistic Regression Functions, Boolean Discriminant Functions.

## 1. Introduction

Over the past several years we have developed various metrics models, including Boolean Discriminant Functions (BDFs) for classifying quality; Kolmogorov-Smirnov distance (K-S test) for estimating metric critical values; various derivative calculations for assessing the quality that could be achieved with various levels of quality control and inspection; stopping rules for deciding how many metrics to use in a discriminate function; point and confidence interval estimates of quality [SCH00]; Relative Critical Value Deviation metrics for indexing

quality; and non-linear regression functions for predicting quality [SCH99]. We have focused this research on the Space Shuttle for which we have a large amount of data.

Now we turn our attention to investigate how logistic regression would compare to our previous methods in classifying quality. Although other researchers have applied logistic regression to classifying quality, our contributions are the following:

- 1) Developed a method to determine the critical values of the Logistic Regression Functions (LRFs) for classifying software quality, using the inverse of the Kolmogorov-Smirnov (K-S) distance;
- 2) Compared the Discriminative Power of LRFs with BDFs;
- 3) Investigated whether LRFs in combination with BDFs would provide better quality discrimination than either one used alone;
- 4) Investigated the ability of LRFs to rank quality; and
- 5) Performed 1 – 4 on a safety critical system: the Shuttle.

We were motivated to do this research for the following reasons – each reason associated with a research question:

- 1) BDFs provide excellent ability to classify low quality modules. However, this result is achieved at a relatively high cost of inspection, where this cost is incurred because modules that are predicted to be fault prone are inspected (see section 4.). Thus, the first research question was: What misclassification and inspection rates could be obtained by using logistic regression?
- 2) The purpose of BDFs is to provide criteria for deciding whether a module is fault prone or not fault prone; BDFs,

as *discrete* functions, do not provide information about the degree to which *individual* modules are fault prone. BDFs only provide the degree of software quality for entire software systems that consist of a set of modules. In the case of the Shuttle, these are *Operational Increments* (OIs). Using the BDF approach, we can compare the quality of software across OIs. Thus, the second research question was: Given that LRFs are the logit of the probability of the occurrence of discrepancy reports (DRs) (reports of deviations between requirements and implementation) on modules as a linear function of key metrics, and thus a *continuous* function, would the LRFs provide additional information about the quality of *individual* modules? See sections 3.2 and 3.3 for details on how BDFs and LRFs are constructed, respectively.

3) In addition to the potential ability of LRFs to *discriminate* quality, we also wanted to investigate their ability to *rank* quality. Thus, the third research question was: Would the array of the number of discrepancy reports written against module  $i$  ( $drcount$ ) rank in approximately the same order as the array of  $P_i$  or logit ( $P_i$ ), where  $P_i$  is the probability of  $drcount > 0$ ?

This paper is organized as follows: review related research; provide an overview of the analysis approach; compare the Discriminative Power of LRFs with BDFs; evaluate the ability of LRFs to rank quality; and conclude with answers to the research questions posed above.

## 2. Related Research

Many models and methods have been employed to classify the quality of software. We briefly describe a selected few that are most related to our research. In two related studies, Briand et al. used logistic regression to assess the probability of a fault in a class as a function of various object-oriented design measures [BRI98, BRI98]. In a similar vein, Tang et al. used logistic regression to investigate the association between object-oriented metrics and fault prone classes [TAN99]. In another example of logistic regression, Khoshgoftaar and Allen used it to classify modules as fault-prone or not fault-prone as a function of faults, requirements, performance, documentation, and software trouble report metrics [KHO97]. They developed a decision rule for deciding whether a module was fault prone based on the costs of misclassification. Khoshgoftaar and Allen also used percentile ranking of quality factors to determine thresholds for identifying the highest priority modules (i.e., the “most fault prone” modules) [KHO98]. This approach has the same objective as our quality ranking method (to be described) except that we use  $P_i$  derived from logistic regression to rank order  $drcount$ . Ohlsson and Wohlin used historical fault distributions and changes

in principal components between releases to classify software fault proneness into one of three categories: “red”, “yellow”, and “green” [OHL98]. This classification is used to indicate the difficulty of maintaining the software. In another approach to classification, Ohlsson and Alberg used Alberg Diagrams to 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 [OHL96]. All of these approaches have a common objective with ours: identify fault prone software. Our work differs in three ways: 1) we investigated whether BDFs in combination with LRFs would improve quality classification of either used alone; 2) we derived a new quality discriminant -- critical values (thresholds) of  $P_i$  that are used to classify quality; and 3) we evaluated the quality ranking ability of LRFs.

## 3. Analysis

Now we evaluate the ability of BDFs and LRFs to classify quality, along with the cost of inspection that would be incurred in achieving this quality. We used an incremental approach to adding metrics to the BDFs and LRFs, as described below [SCH00]. This is accomplished in two activities: *validation* and *application*. *Validation* is an activity that is required in order to identify metrics that can predict low quality software that will require corrective action during *application*. *Application* is an activity during which validated metrics are applied to control software quality. During *validation*, we used Build 1, comprised of 1397 modules (576 with  $drcount > 0$ , 41.23%), as the source of both  $drcount$  and metrics data to compute metrics discriminant functions *retrospectively* (i.e., after both  $drcount$  and metrics data were available). During *application*, we used these discriminant functions and the metrics data from Build 2, comprised of 846 modules (418 with  $drcount > 0$ , 49.41%), to control the quality of this build in real time prior to the availability of  $drcount$  data. In order to not bias the results, Build 2 is neither a descendant of nor chronologically follows Build 1. We then compare the ability of the BDFs and LRFs to classify quality and the inspection cost that would be incurred by the two methods.

These discriminant functions provide a cost effective way of ensuring product quality compared with one hundred percent manual inspection because with the former, the expected quality and associated inspection cost can be quantified, and only those modules that are predicted to be fault prone are subjected to formal manual inspection. Complete manual inspection is infeasible on large systems and is itself fault prone due to the tedium and fatigue that accompanies such an effort. In contrast,

our approach provides automated data collection and computation of module metrics, using a metrics analyzer, and the derivation of discriminant functions, using spreadsheet and statistical tools. Using this approach, the desired quality and the cost of achieving it can be specified in advance and the metrics set and their critical values selected to achieve those objectives.

### 3.1 Incremental Addition of Metrics to Discriminant Functions

We showed in [SCH00] 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 showed that certain metrics are dominant in their effects on classifying quality (i.e., dominant metrics make fewer mistakes in classifying quality than non-dominant ones) and that additional metrics are not needed to accurately classify quality. 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, when 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. Related to the property of *dominance* is the property of *concordance*: the degree to which a set of metrics produces the same result in classifying software quality as metrics are added to the set. A high value of *concordance* implies that additional metrics will not make a significant contribution to accurately classifying quality; hence, these metrics are redundant.

In [SCH100], we developed six BDFs, comprised of one to six metrics in order to evaluate the properties of the BDFs over a range of metrics. As each metric was added to the BDF for Build 1, during the *validation* activity, we noted when the ratio of relative incremental quality to relative incremental inspection (IQIR) reached a maximum. This occurred at three metrics. Adding a fourth metric did not increase the IQIR, although quality continued to increase until the fifth metric was added. On Build 2, we found that the maximum IQIR also occurred at three metrics during the *application* activity. However, in analyzing the LRFs, the criteria of association and model fit were used. The application of these criteria resulted in LRFs with four metrics and six metrics,

respectively. Thus for results to be comparable, we used four and six metric BDFs and LRFs.

### 3.2 Boolean Discriminant Functions

BDFs for OIs are formed by entering module metrics into a Boolean expression in priority order, where the priority is determined by the metrics' K-S maximum distance rank. The maximum distance is equal to the maximum difference between two cumulative distribution functions (CDFs), where the CDFs are distributions of metric values for non-fault prone and fault prone modules. The metrics' critical values are obtained from the inverse of the K-S maximum distance. This is the value of a metric that corresponds to the maximum distance. A BDF is a Boolean function consisting of *AND* and *OR* operators, module metric values  $M_{ij}$ , and metric critical values  $MC_j$  that is used to classify modules as either *accepted* or *rejected* for module  $i$  and metric  $j$ . When the following BDF evaluates to *true*, modules are *accepted*:

$$(M_{i1} \leq MC_1) \dots \div (M_{ij} \leq MC_j) \dots \div (M_{im} \leq MC_m)$$

When the following BDF evaluates to *true*, modules are *rejected*:

$$(M_{i1} > MC_1) \dots \equiv (M_{ij} > MC_j) \dots \equiv (M_{im} > MC_m)$$

With the addition of quality factor values ( $F_i$ ) and quality factor critical value  $FC$ , as shown in equation (1), we obtain module counts that are used to classify modules into one of four categories, as defined below.

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

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;  $\text{COUNT}(0) = 0$ . In our examples,  $F_i$  is *drcount* and  $FC = 0$ .

These counts have the following interpretations:

$C_{11}$  = The count of modules in the build that are high quality and no metric value exceeds its critical value.

$C_{12}$  = The count of modules in the build that are high quality and one or more metric values exceed its critical value.

$C_{21}$  = The count of modules in the build that are low quality and no metric value exceeds its critical value.

$C_{22}$  = The count of modules in the build that are low quality and one or more metric values exceed its critical value.

The counts  $C_{11}$  and  $C_{22}$  are correct classifications and the counts  $C_{12}$  and  $C_{21}$  are misclassifications. These counts are inputs to the remaining computations. Metric critical values provide a threshold between two levels (e.g., *high* and *low*) of the quality of the software [SCH00]. The critical values derived from applying the K-S method are shown in Table 1. Metrics were entered incrementally in the BDFs in the sequence given by the K-S ranks in Table 1.

### 3.3 Logistic Regression Functions

Logistic regression is based on the concept of the success probability  $P_i$  of a binary event (e.g., the probability of  $drcount > 0$  on module  $i$ ). The odds of such an event are defined as the probability of success divided by the probability of failure, or  $P_i / (1 - P_i)$ . A logarithmic transformation is applied to this quantity to produce  $\text{Logit}(P_i)$  (see equation (2)). This transformation maps  $P_i$  from the range (0,1) to the range  $(-\infty, \infty)$ : an unconstrained continuous range.  $\text{Logit}(P_i)$  is then modeled as a linear function of independent variables (e.g.,  $M_{ij}$ ), as in equation (2) [COL91]. We formulated the logistic regression equations as the logit of the probability of the occurrence of discrepancy reports on a module as a linear function of several metrics. The following is the equation obtained from binary logistic regression that predicts the probability  $P_i$  of  $drcount > 0$  on module  $i$ , using four metrics:

$$\text{Logit}(P_i) = \log [P_i / (1 - P_i)] = (-1.813) + (0.012225 * C) + (0.012381 * S) - (0.00148 * E2) - (0.0034733 * L) \quad (2)$$

Binary logistic regression fits a model with one or more predictors using an iterative-reweighted least squares algorithm to obtain maximum likelihood estimates of the model parameters [Col91]. Equation (2) was formulated based on the number of metrics that had the maximum association of 87.2% (i.e., the maximum association between the predicted and actual data). Metrics were entered in equation (2) in the sequence given by the K-S ranks in Table 1.

The following is the equation obtained from binary logistic regression that predicts the probability of  $DRs > 0$  on module  $i$ , using six metrics:

$$\text{Logit}(P_i) = (-2.7003) + (0.010344 * C) + (0.002841 * S) + (0.00048E2) - (0.0020148 * L) + (0.06908 * E1) + (0.002876 * N) \quad (3)$$

As in equation (2), metrics were entered in equation (3) in the sequence given by the K-S ranks in Table 1. This equation had better fit than equation (2):  $\Pi^2/df = 1.16$  versus 1.50 but a lower association of 86.7%.

A logit equation by itself does not provide a discriminant of quality; a criterion for deciding whether a module is fault prone or not fault prone is required [KHO97]. We used the inverse of the K-S distance [SCH00]. This was accomplished by finding the value  $PC$  of  $P_i$ , where there was the maximum distance between the cumulative distribution function (CDF) for  $drcount = 0$  versus  $drcount > 0$  (i.e., the critical value). These values were  $PC = .00014828$  for equation (2) and  $PC = .00012605$  for equation (3). An example is shown for the four metric case in Figure 1, and Figure 2 shows the plot of  $drcount$  versus  $P_i$ ; both figures indicate the critical value of  $P_i$ . When these discriminants were used to classify quality for Build 1, the results in Table 2 were obtained.

The module counts given by equation (1) can be applied to the LRFs by replacing  $M_{ij}$  with  $P_i$  and  $MC_j$  with  $PC$ .

The module counts then have the following interpretations:

$C_{11}$  = The count of modules in the build that are high quality and  $P_i$  does not exceed its critical value.

$C_{12}$  = The count of modules in the build that are high quality and  $P_i$  exceeds its critical value.

$C_{21}$  = The count of modules in the build that are low quality and  $P_i$  does not exceed its critical value.

$C_{22}$  = The count of modules in the build that are low quality and  $P_i$  exceeds its critical value.

### 3.4 Misclassification

We computed the degree of misclassification by noting that ideally  $C_{11} = N_1 = n_1$ , the number of *accepted* modules and the number of high quality modules, respectively,  $C_{12} = 0$ ,  $C_{21} = 0$ , and  $C_{22} = N_2 = n_2$ , the number of *rejected* modules and the number of low quality modules, respectively. 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, where  $n$  is the number of modules in the build:

$$\text{Proportion of Type 1: } p_1 = C_{21}/n \quad (4)$$

$$\text{Proportion of Type 2: } p_2 = C_{12}/n \quad (5)$$

### 3.5 Quality

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

$$\text{LQC: Proportion of low quality modules (e.g., } drcount > 0) \text{ correctly classified} = C_{22}/n_2 \quad (6)$$

Second, we estimated the ability of the metrics to correctly classify quality, given that the BDF or LRF has classified modules as acceptable. This is done by summing quality factor in the accept category to produce Remaining Factor, RF (e.g., remaining quality factor *drcount* or number of DRs that were not caught when classifying quality), given by equation (7).

$$RF = \sum_{i=1}^n F_i \text{ FOR } ((F_i > 0) \wedge (M_{i1} \leq MC_1) \dots \wedge (M_{ij} \leq MC_j) \dots \wedge (M_{im} \leq MC_m))$$

for  $j = 1, \dots, m$  (7)

This is the sum of  $F_i$  (e.g., *drcount*) on modules incorrectly classified as high quality because  $(F_i > FC) \div (M_{ij} \# \square MC_j)$  for these modules. Again, equation (7) can be applied to the LRFs by replacing  $M_{ij}$  with  $P_i$  and  $MC_j$  with  $PC$ .

We estimated the *Quality* ( $Q$ ) of the build by equation (8), where  $TF$  is the total  $F_i$  for the build.

$$Q = (TF - RF) / TF \quad (8)$$

Assuming that the problems reported in the DRs of the rejected modules are corrected, equations (6) and (8) give the quality that *would* have been achieved *if* the discriminant functions had been used in the design phase of Build 1 to reject low quality modules. Furthermore, these equations predict the quality that would be achieved on Build 2.

### 3.6 Inspection

We were interested in weighing the cost of inspection requirements (i.e., percent of modules rejected and subjected to detailed inspection) against the quality that is achieved, for various BDFs and LRFs. We estimated inspection requirements by noting that all modules that are rejected must be inspected; this is the count  $C_{12} + C_{22}$ .

Thus, the proportion of modules that must be inspected is given by:

$$I = (C_{12} + C_{22}) / n \quad (9)$$

### 3.7 Tradeoff between Quality and Inspection

Note that equation (8) classifies quality according to correctness of *drcount* classification, whereas equation (6) classifies quality according to correctness of *module* classification. Therefore, it is appropriate to evaluate the effectiveness of a discriminant with two types of quality/inspection ratio, as in equations (10) and (11), respectively.

$$QIR = Q/I \quad (10)$$

$$LIR = LQC/I \quad (11)$$

For a safety critical system, it would be appropriate to emphasize quality, using  $Q$  and  $LQC$  as the criteria; for a commercial system, the quality/inspection ratios might be more appropriate. However, we caution that a policy that trades quality for decreased costs during development could be short sighted. The costs of customer ill will and the cost of correcting problems could outweigh reductions in development cost. The results of computing equations (4)-(6) and (8)-(11) are shown for BDFs and LRFs for four and six metrics in Table 2 for Build 1.

## 4. Comparison of BDFs with LRFs

### 4.1 Validation Predictions

Table 2 provides a comparison of the ability of BDFs and LRFs to classify quality and the inspection cost that would be incurred in achieving this quality during the *validation* activity using Build 1. We see that BDFs are superior quality classifiers but have a higher inspection cost and lower quality/inspection ratios. Given these results, it occurred to us that combining a BDF with a LRF would yield even higher quality and lower inspection cost than either one used alone. In order to achieve this goal, we used the following modified BDFs that form the union of the original BDF and the LRF conditions for *accepted* and *rejected* modules, respectively:

$$((M_{i1} \leq MC_1) \dots \div (M_{ij} \leq MC_j) \dots \div (M_{im} \leq MC_m)) \equiv (P_i \leq PC)$$

$$((M_{i1} > MC_1) \dots \equiv (M_{ij} > MC_j) \dots \equiv (M_{im} > MC_m)) \equiv (P_i > PC)$$

Using the BDF as the baseline, this approach permits the LRF to accept a certain number of high quality modules that were rejected by the BDF and to reject a certain number of low quality modules that were accepted by the BDF. The results are shown in the last row of

Table 2 for four metrics, where the quality classification has improved. This approach would be appropriate for a safety critical system but might not be worth the additional cost of analysis for a commercial system.

## 4.2 Application Results

Table 3 shows the results obtained by applying the BDFs and LRFs validated in Build 1 to Build 2. The critical values of BDFs and LRFs,  $MC_j$  and  $PC$ , respectively, obtained from Build 1, were used with the metrics  $M_{ij}$  of Build 2 during its design phase to accept modules as not fault prone or to reject modules as fault prone. This was accomplished in the *absence* of *drcount* data. This was the case because in the actual application of the validated BDFs and LRFs, complete *drcount* data would not be available until the end of the test phase of Build 2. The major point of validation is to develop discriminant functions that can predict quality and inspection cost of the *application* product sufficiently early to detect and correct problems when the cost of correction is relatively low. However, we *retrospectively* analyzed the accuracy of prediction by using the critical values from Build 1 and the metrics data and *drcount* from Build 2 and recomputed equations (1)– (11).

We see in Table 3 that the quality achieved by Build 2 BDFs and LRFs is slightly higher but the cost of inspection is higher (i.e., more rejected modules) than that predicted during Build 1. Such an outcome would be favorable for safety critical systems but might be considered too costly for commercial systems. As in the case of Build 1, improved quality is obtained over either the BDF or LRF alone, by combining the two functions. Compared to the predictions of Build 1, the quality achieved is slightly lower than that predicted in Build 1 and the cost of inspection is higher.

## 5. Ability of LRFs to Rank Quality

### 5.1 Validation Predictions

We tested the ability of the LRFs to rank quality (i.e., the degree of association between  $P_i$  and *drcount*) during the *validation* activity using Build 1. We did this evaluation by using the *Mann-Whitney* test [CON71]. We used this test to do a two-sample rank test for the difference between the population medians of  $P_i$  and *drcount*, where  $H_0$ : medians of  $P_i$  and *drcount* are equal and  $H_1$ : medians are not equal. The results are shown in Table 4, which indicates there is *not* a statistically significant difference between the medians of  $P_i$  and *drcount* at the indicated for both the four and six metric LRFs. We performed an additional analysis to determine

the percentage of *drcount* corresponding to the highest and lowest 100 ranks of  $P_i$ , using Build 1. The predictions are shown in Table 4 for the four and six metric cases for Build 1. Figure 3 shows *drcount* versus the rank of  $P_i$  for the four metric LRF for Build 1.

The purpose of this analysis was to establish a predictor threshold (i.e., *highest* 100 ranks of  $P_i$ ) of the *lowest* quality modules in the *Application* product (Build 2). That is, after calculating  $P_i$  for Build 2, ranking them and determining the 100 *highest* rank threshold, all modules within the 100 *highest* rank range were identified as highly fault prone. This was accomplished in the absence of *drcount* data for Build 2. Similarly, the 100 *lowest* ranks of  $P_i$  were used to establish the threshold for modules that are *not* highly fault prone

### 5.2 Application Results

The four metric and six metric LRFs on Build 2 failed the *Mann-Whitney* test (i.e., relatively large  $\alpha$  ), as shown in Table 4. However, we made a *retrospective* analysis of Build 2, using the quality ranking thresholds identified in Build 1 and the *drcount* of Build 2. The analysis of the highest and lowest 100 ranks for both the four and six metric cases indicates a close correspondence between predictions and results, as shown in Table 4. Figure 4 shows *drcount* versus the rank of  $P_i$  for the four metric LRF for Build 2.

## 6. Conclusions

Referring to the research questions we posed in the Introduction, we arrive at the following conclusions, based on the analysis and results documented herein:

1) What misclassification and inspection rates could be obtained by using logistic regression? We found that for the same software system and using the same set of metrics, BDFs were still superior to LRFs for quality discrimination.

2) Would the LRFs provide additional information about the quality of *individual* modules? We found that LRFs used in isolation were of limited value. However, when combined with BDFs they provided a marginal improvement in quality discrimination for low quality modules. This is the case because the quality discrimination ability of BDFs is already high. However, when LRFs are added, inspection cost is reduced from that incurred when BDFs are used alone. We consider this a significant finding in terms of providing an accurate quality predictor (1.25% error;  $Q=98.75\%$  for BDF\LRF in Table 3) for safety critical systems at reasonable cost

(relatively high values of QIR and LIR for BDF\LRP in Table 3). This is the lowest prediction error rate we have found in the literature.

3) Would the array of the number of discrepancy reports (reports of deviations between requirements and implementation) written against a module (*drcount*) rank in approximately the same order as the array of  $P_i$  or logit ( $P_i$ ), where  $P_i$  is the probability of  $drcount > 0$ ? We found that the ranking of  $P_i$  provided accurate thresholds for identifying both low and high quality modules. However, the statistical test of difference in ranks between  $P_i$  and *drcount* yielded mixed results.

The method we developed for determining the critical value of LRFs, using the inverse of the Kolmogorov-Smirnov (K-S) distance, provided good balance between quality and inspection cost.

We are confident about the ability of BDFs to consistently provide high accuracy quality classification for the Shuttle software because prior studies involved three builds and four subsets of these builds, where the validated BDFs yielded high accuracy (approximately 3% error) across builds without revalidating [SCH00, SCH100]. However, although 2,244 modules were used in the current study, the analysis to date on LRFs involved only two builds. The results are encouraging but more builds should be analyzed to increase confidence in the results. The methods we have presented are general and not particular to the Shuttle. Thus, the methods should be applicable to other domains. However, the metric set, critical values, and numerical results would in general differ from those used for the Shuttle.

## 7. Acknowledgements

We wish to express our appreciation to the following: Dr. Allen Nikora of the Jet Propulsion for the support of this project, Dr. John Munson of Cylant Technology for providing the data, and to the anonymous reviewers.

## 8. References

[BRI98] Briand, Lionel C., John Daly, Victor Porter, and Jürgen Wüst, "Predicting Fault-Prone Classes with Design Measures in Object-Oriented Systems", Proceedings of the Ninth International Symposium on Software Reliability Engineering, IEEE Computer Society, Los Alamitos, CA, November 1998, pp. 334-343.

[BRI98] Briand, Lionel C., John Daly, Victor Porter, and Jürgen Wüst, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems", Proceedings of the Fifth International Metrics Symposium, IEEE Computer Society, Los Alamitos, CA, November, 1998, pp. 246-257.

[COL91] D. Collett, *Modelling Binary Data*, Chapman & Hall, 1991.

[CON71] W. J. Conover, *Practical Nonparametric Statistics*, John Wiley & Sons, Inc., 1971.

[KHO97] Khoshgoftaar, Taghi, M. and Edward B. Allen, "Logistic Regression Modeling of Software Quality", TR-CSE-97-24, Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, FL., March 1997.

[KHO98] Khoshgoftaar, Taghi, M. and Edward B. Allen, "Predicting the Order of Fault-Prone Modules in Legacy Software", Proceedings of the Ninth International Symposium on Software Reliability Engineering, IEEE Computer Society, Los Alamitos, CA, November 1998, pp.344 -353.

[OHL96] Ohlsson, Niclas and Hans Alberg (1996), "Predicting Fault-Prone Software Modules in Telephone Switches", *IEEE Transactions on Software Engineering*, 22, 12, 886-894

[OHL98] Magnus C. Ohlsson and Claes Wohlin, "Identification of Green, Yellow and Red Legacy Components", Proceedings of the International Conference on Software Maintenance, IEEE Computer Society, Los Alamitos, CA November 1998, pp. 6-15.

[SCH99] Norman F. Schneidewind and Allen P. Nikora, "Predicting Deviations In Software Quality By Using Relative Critical Value Deviation Metrics", Proceedings of The 10<sup>th</sup> International Symposium on Software Reliability Engineering, IEEE Computer Society, Los Alamitos, CA , November , 1999, pp. 136-146.

[SCH00] 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.

[SCH100] Norman F. Schneidewind, "On the Repeatability of Metric Models and Metrics Across Software Builds", Proceedings of the Eleventh International Symposium on Software Reliability Engineering, IEEE Computer Society Los Alamitos, CA, October, 2000. (to be published).

[TAN99] Mei-Huei Tang., Ming-Hung Kao, and Mei-Hwa Chen "An Empirical Study on Object-Oriented Metrics", Proceedings of the Sixth International Metrics Symposium, IEEE Computer Society, Los Alamitos, CA, November, 1999, pp. 242-249.

+  
-

Table 1: Kolmogorov-Smirnov Distance for $drcount=0$ vs. $drcount>0$ Validation: Build 1 (n=1397 modules)					
Metric (symbol)	Definition (counts per module)	Critical Value	Distance	$\alpha$	Rank
prologue size (C)	change history line count in module listing	63	0.592	0.005	1
statements (S)	executable statement count	27	0.505	0.005	2
eta2 (E2)	unique operand count	45	0.472	0.005	3
loc (L)	non-commented lines of code count	29	0.462	0.005	4
eta1 (E1)	unique operator count	9	0.430	0.005	5
nodes (N)	node count (in control graph)	17	0.427	0.005	6

Table 2: BDF – LRF Comparison, <i>Validation Predictions</i> (Build 1, n=1397 modules)								
Method		p <sub>1</sub> (%)	p <sub>2</sub> (%)	LQC (%)	Q (%)	I (%)	QIR	LIR
	Metrics							
BDF	C, S, E2, L	2.00	29.35	95.14	98.22	68.58	1.43	1.38
BDF	C, S, E2, L, E1, N	1.43	34.93	96.53	98.76	74.73	1.32	1.29
LRF	C, S, E2, L	5.25	18.33	87.19	94.73	54.06	1.75	1.61
LRF	C, S, E2, L, E1, N	4.02	21.68	90.21	94.57	58.72	1.61	1.54
BDF $\equiv$ LRF	C, S, E2, L	0.72	17.32	98.26	99.53	57.84	1.72	1.70

Table 3: BDF – LRF Comparison, <i>Application Results</i> (Build 2, n=846 modules)								
Method		p <sub>1</sub> (%)	p <sub>2</sub> (%)	LQC (%)	Q (%)	I (%)	QIR	LIR
	Metrics							
BDF	C, S, E2, L	1.30	36.29	97.37	98.43	84.40	1.33	1.32
BDF	C, S, E2, L, E1, N	0.59	41.84	98.80	99.00	90.66	1.09	1.09
LRF	C, S, E2, L	3.07	27.66	93.69	97.19	73.90	1.32	1.27
LRF	C, S, E2, L, E1, N	2.73	30.69	94.47	97.33	77.71	1.25	1.22
BDF $\nabla$ LRF	C, S, E2, L	0.95	23.88	98.09	98.75	72.34	1.37	1.36

Table 4: LRF Quality Rankings, <i>Validation Predictions</i> (Build 1, n=1397 modules) vs. <i>Application Results</i> (Build 2, n=846 modules)				
	Metrics	Mann Whitney	High 100 Ranks $drcount$ (%)	Low 100 Ranks $drcount$ (%)
Prediction	C, S, E2, L	.0000	40.93	.16
Result	C, S, E2, L	.5545	46.99	.67
Prediction	C, S, E2, L, E1, N	.0000	43.51	1.59
Result	C, S, E2, L, E1, N	.6434	47.61	1.02

Figure 1: CDFs vs.  $P_i$ , Logistic Regression (4 Metrics, Build 1)

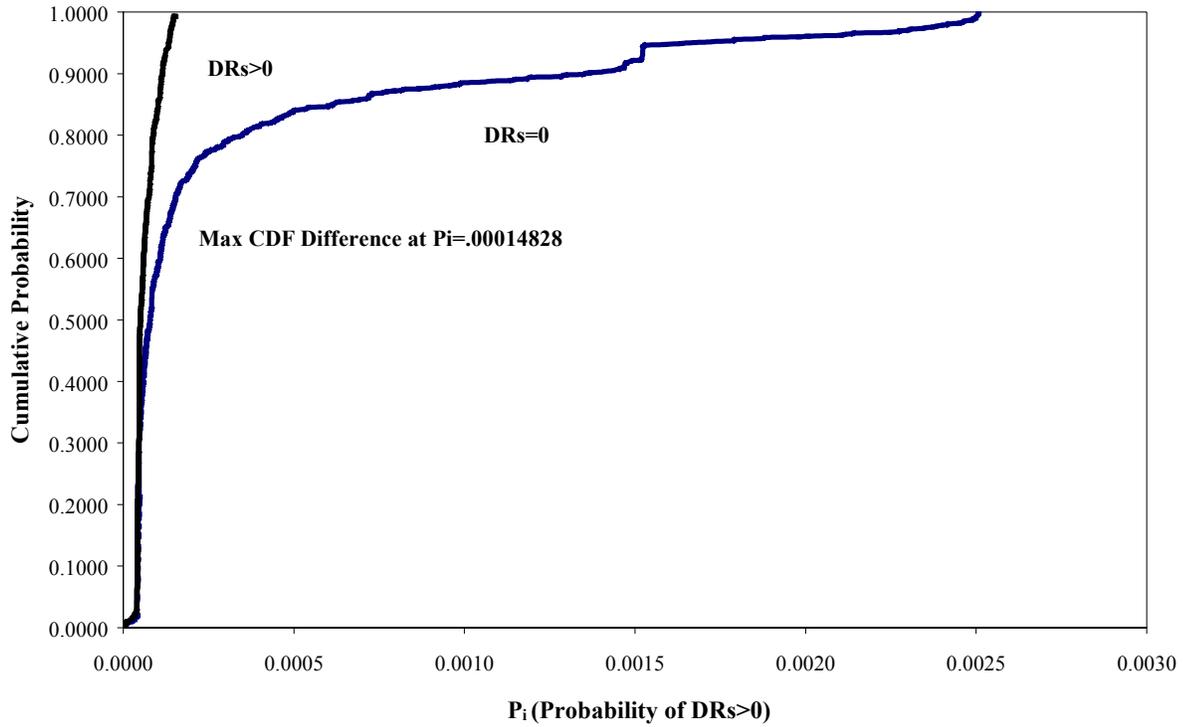


Figure 2: drcount vs.  $P_i$ , Logistic Regression (4 Metrics, Build 1)

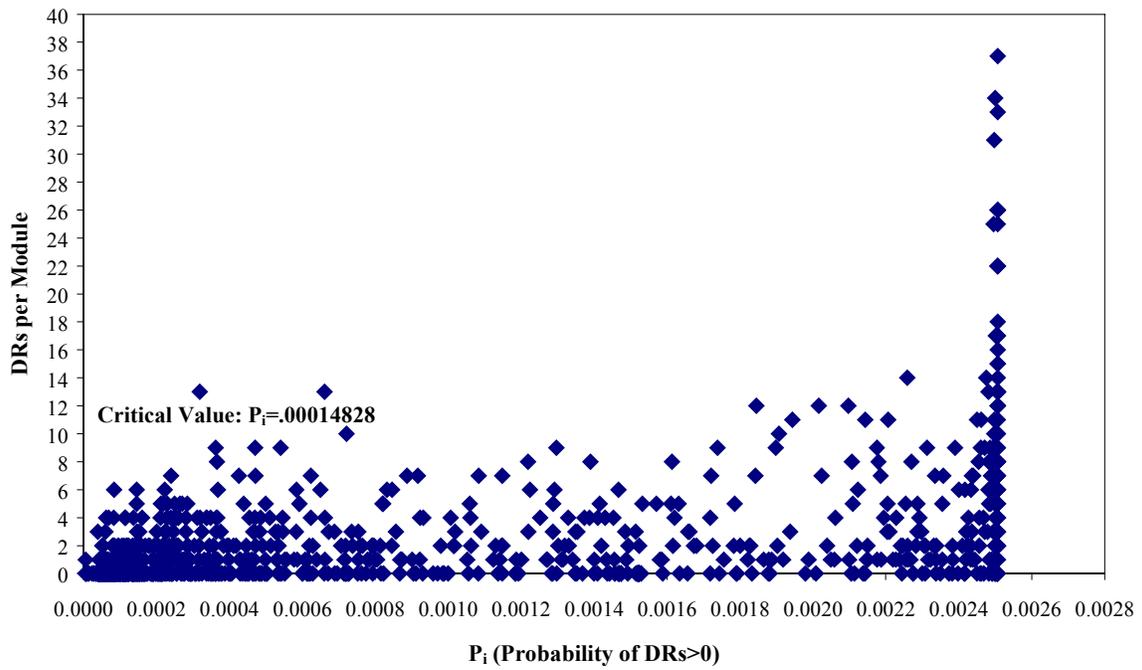


Figure 3: drcount vs. Rank of  $P_i$ , Logistic Regression (4 Metrics, Build 1)

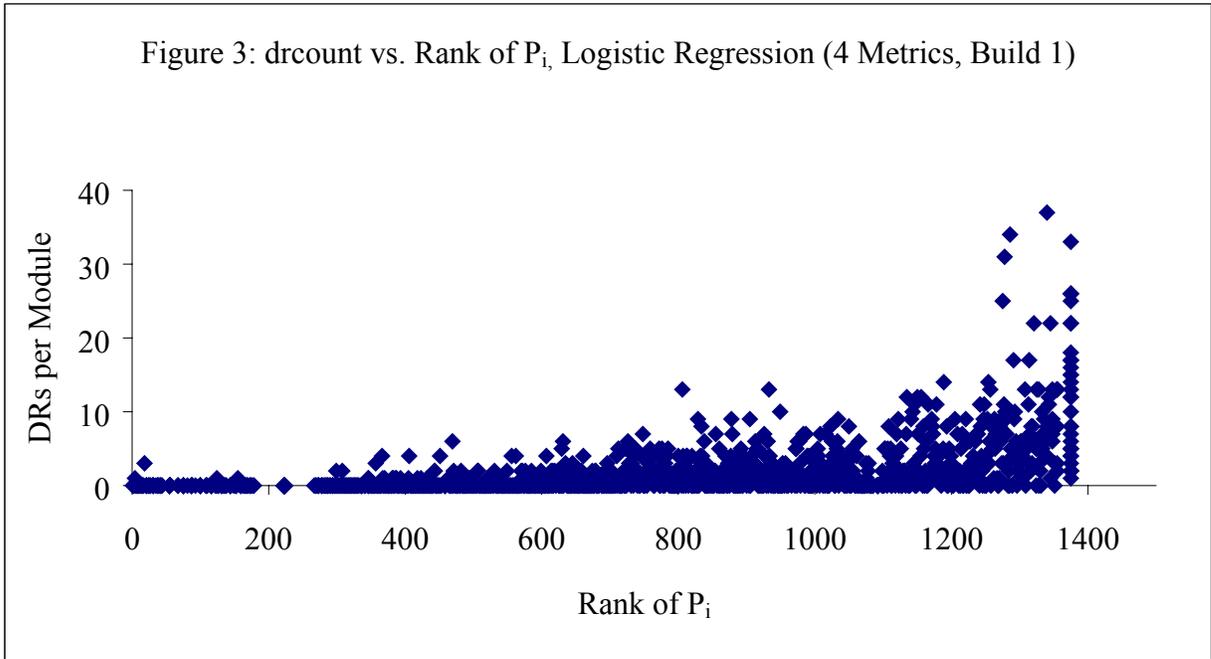


Figure 4: drcount vs Rank of  $P_i$ , Logistic Regression (4 Metrics, Build 2)

