ELSEVIER

The Journal of Systems and Software 67 (2003) 153–163

The Journal of Systems and Software

www.elsevier.com/locate/jss

An empirical comparison and characterization of high defect and high complexity modules

A. Güneş Koru, Jeff Tian *

Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122, USA

Received 6 February 2002; received in revised form 20 April 2002; accepted 10 May 2002

Abstract

We analyzed a large set of complexity metrics and defect data collected from six large-scale software products, two from IBM and four from Nortel Networks, to compare and characterize the similarities and differences between the high defect (HD) and high complexity modules. We observed that the most complex modules often have an acceptable quality and HD modules are not typically the most complex ones. This observation was statistically validated through hypothesis testing. Our analyses also indicated that the clusters of modules with the highest defects are usually those whose complexity rankings are slightly below the most complex ones. These results should help us better understand the complexity behavior of HD modules and guide future software development and research efforts.
© 2002 Elsevier Inc. All rights reserved.

Keywords: Software quality; Complexity metrics; Complexity-defect association; Risk identification; Hypothesis testing

1. Introduction

Software quality is of critical importance to many software products. It can usually be directly characterized by various defect related measures, with high quality usually associated with low numbers of defects, and vice versa. However, defects cannot be measured ahead of time. Consequently, various internal properties of products are measured and historical data are used in various models to predict quality.

Complexity, as an internal property of software products, can be measured using many techniques applied on source code, design, and various other software artifacts (Card and Glass, 1990; Halstead, 1977; McCabe, 1976). It is commonly observed and intuitively believed that a positive correlation exists between complexity and defect count. In addition, although the defect-failure relationship is not straightforward, the common intuition is that high complexity (HC) also leads to high number of failures, thus having a negative effect on quality and reliability (Munson and Khoshgoftaar, 1992).

Studies have also shown that the majority of problems and rework in software development is caused by a small proportion of the modules (Fenton and Ohlsson, 2000; Porter and Selby, 1990; Tian and Troster, 1998). Prediction of these problem prone modules can lead to a substantial improvement in software quality. In the prerelease stage, the prediction of problem prone modules is usually attempted by using complexity and other measures as predictors. Typically, the measurement characteristics of historically high defect (HD) modules are used in new releases to identify and characterize problem prone modules. Consequently, it is critically important to understand the measurement characteristics of these HD modules.

In this paper, we aim to characterize and compare the HC and HD modules in order to understand their similarities and differences. This understanding will not only help us check the validity of the commonly assumed positive correlation between complexity and defects as applied to HC and HD, it will also help us select or develop more effective risk identification techniques for quality assurance and improvement.

* Corresponding author. Tel.: +1-214-768-2861; fax: +1-214-768-3085.

E-mail addresses: gkoru@engr.smu.edu (A. Güneş Koru), tian@engr.smu.edu (J. Tian).

0164-1212/03/$ - see front matter © 2002 Elsevier Inc. All rights reserved.
doi:10.1016/S0164-1212(02)00126-7

The pre-release defects (named as *defects* hereafter) and complexity data we analyzed in this paper belong to six large scale software products, two from IBM and four from Nortel Networks. Once we characterize HC and HD clusters on these products, we use hypothesis testing to make statistical inferences about the sameness of them. First, we set forth a null hypothesis (also called default hypothesis), which states that the two samples are identical. We also set an alternative hypothesis that states the opposite of it. After applying appropriate statistical techniques, if enough evidence can be found to reject the null hypothesis, we reject it safely in favor of the alternative hypothesis. In addition to hypothesis testing, we analyze top defect clusters further in order to understand their place in the complexity ranking.

In this paper, we start by briefly describing the products of our study, the data that belong to them, and the metrics in Section 2. Section 3 discusses the identification of HC and HD clusters. The hypothesis testing in Section 4 gives our results about the sameness of the HC and HD clusters. In Section 5, we examine the complexity rankings of the top defect clusters to provide some useful information about their measurement characteristics. Finally, Section 6 presents our discussions, conclusions, and perspectives. Appendices give further detailed information about the metrics used, and tree-based modeling technique employed to identify the HC and HD clusters.

2. Products, metrics, and data

Each product we examined includes a number of modules designed and implemented to perform a specific function within a large software system. A module in our study corresponds to a software piece that includes smaller pieces usually named as functions or subroutines. The average size of a module is around one thousand lines of code. Associated with these modules are the measurement results using various metrics. They were originally selected for quality improvement purposes in their respective organization. Here, we utilize all of the available data collected using these broad range of metrics in order to make more generalizable conclusions. Hereafter, we will refer to them generically as defect and complexity metrics. All the measurement results of a module that are obtained using these metrics and the module identity information are together considered as a single observation and taken as a single data point in our study.

2.1. IBM products

The first IBM product we analyzed is a legacy system, labeled IBM-LS, and the second one is a new system, labeled IBM-NS. IBM-LS and IBM-NS are relational database management systems produced in the IBM Software Solutions Toronto Laboratory. Each of them contains about one million lines of code. IBM-LS contains 1302 modules and IBM-NS contains 995 modules. Each module has a number of procedures. IBM-LS was implemented in PL/AS, which is a PL/I-like programming language; IBM-NS was implemented in C and C++ programming languages.

As a quality measure, defect count is kept for each module. The metrics for IBM-LS and IBM-NS include design metrics based on Card and Glass (1990), size metrics (lines of code and lines of comments), change metrics (absolute and relative size change from previous release to the current release), and complexity metrics (McCabe's cyclomatic complexity (McCabe, 1976), software science family of metrics (Halstead, 1977), etc.). IBM-LS was measured by 15 metrics and 11 of them were also used in IBM-NS, with the help of existing tools and other utility programs. More discussion about these products, metrics used, and related predictive quality models can be found in Tian and Troster (1998). More information and a complete list of these metrics can be found in Appendix A.

2.2. Nortel Networks products

The Nortel Networks products analyzed in this paper are named as NT-1, NT-2, NT-3 and NT-4, which are telecommunications software. The sizes of them are around one million lines of code each. There are 804, 1098, 712, and 900 modules in NT-1, NT-2, NT-3, and NT-4 respectively. These products were implemented in Protel, a programming language with a Pascal-like syntax.

Similar to the IBM products, pre-release defect count is taken as a measure of quality. Other data were collected by the tool, early risk assessment of latent defects (EMERALD) (Hudepohl et al., 1996b), a software risk assessment tool first released in 1992 by Nortel. EMERALD uses a source code analyzer to obtain the static metrics related to code. The metrics for the Nortel Networks' products we studied form a more extensive set, including 49 metrics of volume, testability, decision complexity, independent path, structuredness, dead code, readability, and section dependability, described in Hudepohl et al. (1996a) and Tian et al. (2001). A complete list and more information about these metrics can be found in Appendix B.

3. Identification of HC and HD clusters

In practical applications, it is common to have considerable data fluctuations due to product and development process variations and various other factors, which may lead to unstable observations and modeling

results. A large number of observations and properly grouped data clusters can generally reduce such fluctuations, and produce models with results more likely to be generalizable to a wider variety of environments. Because of this, we base our study on the HC and HD clusters instead of single observations of highest defect or highest complexity modules.

3.1. HD and HC modules

Each module of a software product corresponds to one single observation in our study. To test our hypothesis about HD and HC modules, we need to first identify the observation data associated with these two types of modules. There are two ways to identify such sets of modules for any product:

- *Independent identification:* We can rank the modules by their defect counts, and classify the top defect modules (either as a fixed number of modules, say N, or as a percentage of all modules) into a set $\mathrm{Top}\,D$. Modules with HC according to a given metric m can be identified similarly as $\mathrm{Top}\,C_m$. However, since we have multiple complexity metrics, we can tally their individual $\mathrm{Top}\,C_m$ sets into a union set $\mathrm{Top}\,C$. We do not need a similar union set for $\mathrm{Top}\,D$.
- *Co-identification of both sets:* Since complexity metrics are often used as predictors of defect count, we can cluster modules according to their actual defect count, using its complexity as the dividing factor. This technique is described next.

3.2. Clustering individual modules into HD and HC sets

To study the relationship between defect counts and complexity, particularly to characterize HD and HC modules, we can cluster modules with similar defect counts and similar complexity measurement results together. The grouping of these individual data points into clusters can be carried out using various statistical analysis techniques for clustering (Venables and Ripley, 1994). In this case, we have a single response variable (defect count), and a single predictor variable (complexity metric). A simplified clustering algorithm using tree-based models (Clark and Pregibon, 1993) supported by a commercial tool S-PLUS [1] can be used. Another reason that we use tree-based models is the opportunity to integrate our analysis with our tree-based defect and reliability models (Tian, 1995; Tian and Troster, 1998) for risk identification and quality improvement.

A brief summary of tree-based modeling technique, its algorithm (Fig. 5), and an example (Fig. 6) can be found in Appendix C. When using tree-based models to find data clusters of similar defect counts and complexity measurement results, we start with the complete set of data, and recursively partition it into two smaller subsets. For a given product, conditions defined on the value v_m according to a complexity metric m, and a cutoff value c in the form of $v_m < c$, or $v_m \geqslant c$, defines a binary partition (see Fig. 5, Step 3). Each recursive partitioning selects a cutoff value c to minimize the difference between the predicted defect count (in this case, it is the average defect count) and the actual defect counts for individual data points in the partitioned subsets. The selected c value should result in such two disjoint subsets that the sum of the deviances of defect counts in these subsets is the minimum among all possible subset pairs that other cutoff values could create.

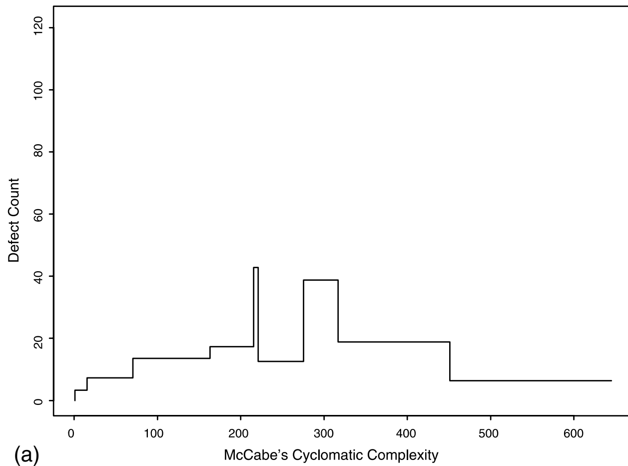
Consequently, these partitions derived from the corresponding tree-based model give us a series of segments associated with the leaf nodes of the tree. Neighboring data points with similar defect counts are grouped into the same complexity segment, where the segment is defined by a pair of upper and lower bounds of their measured complexity according to the given complexity metric m. Data points from different neighboring segments have different defect counts and different measured complexity. The entire complexity range is partitioned into unequal groups such that the defect values is close for modules that fall into each group.

When represented graphically, the above partitions or data clusters at the leaf nodes of the tree can be interpreted and visualized as a piece-wise linear model, maintaining a constant defect count that is the average for each complexity segment. Fig. 1a is an example of such a model for IBM-NS. The defect counts are plotted against McCabe's cyclomatic complexity and our model is shown as a piece-wise linear curve. Fig. 1b is the scatter plot of the original data, where each point represents one module. When Fig. 1a is overlaid with Fig. 1b, it can be seen that each piece of our model represents a constant defect count, which is the average defect count of the data points in that piece's complexity range. The corresponding tree-based model to the piece-wise linear model in Fig. 1a is given in Fig. 6.

Two clusters of interest to our hypothesis testing and risk identification can be identified from such piece-wise linear models according to the complexity metric m:

- The right most segment corresponding to HC_m, the cluster with the highest complexity, where the actual defect counts conform relatively closely to the average defect count represented by the height of the rightmost horizontal line segment corresponding to this cluster.
- The cluster with the highest average defect count can be identified as HD_m, which can be characterized by the highest line segment and the associated segment cutoff complexity values.

[1] S-PLUS is a trademark of Insightful.



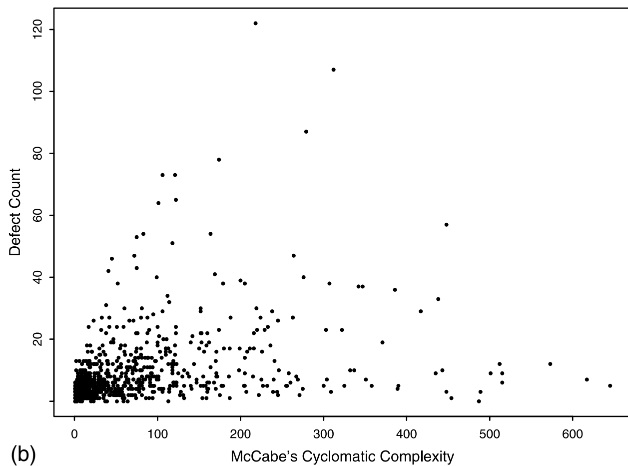


Fig. 1. (a) A piece-wise linear model relating defect count to complexity (Type B2) for IBM-NS and (b) the scatter plot of the original data.

With this interpretation, our hypothesis testing then can be reduced to testing the difference between the two sets HC_m and HD_m. If the two sets do not coincide or they are statistically distinguishable, then our null hypothesis introduced in Section 1 can be rejected safely.

Because of our particular focus on HC_m and HD_m, piece-wise linear models are appropriate for our purpose. In Fig. 1b, it can be noted that there are so many data points clustering close to the origin that individual data points might become obstructed. This does not detract from the appropriateness of the piece-wise linear models since these data points and their area in the plot are irrelevant to our HC_m and HD_m. In our further examples, we do not depict the scatter plots since their corresponding piece-wise linear models are explanatory enough.

3.3. Preliminary analysis of HC and HD

In order to interpret our results, we used the following classification on our piece-wise linear models:

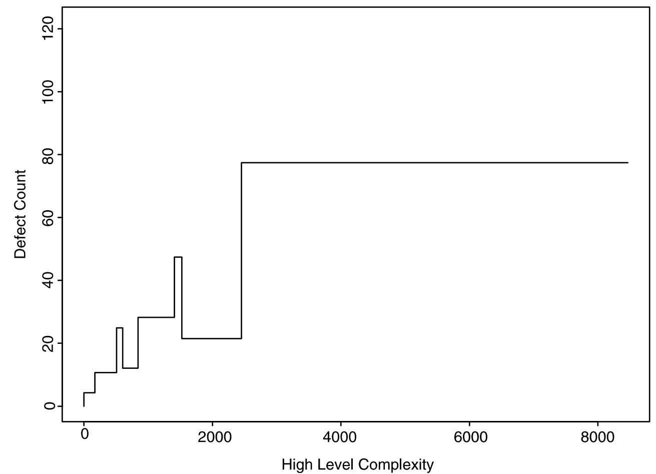


Fig. 2. Type A piece-wise linear model for IBM-NS.

- Type A: HC and HD clusters are identical and appear as the right most cluster having the highest defect count, such as in Fig. 2.
- Type B: HD cluster precedes the HC cluster, such as in Fig. 3.

Each Type A model would support our null hypothesis, namely, HC and HD clusters are the same. On the other hand, each Type B model, provided that the defect distribution between the HC and HD clusters are different (to be statistically tested later in Section 4), rejects our null hypothesis in favor of the alternative hypothesis.

Table 1 lists the metrics that result in Type A and Type B models for each product. The complete names and descriptions of these metrics can be found in Appendices A and B. None of the metrics consistently resulted in Type A models across different products, which would imply our null hypothesis to be true for this set of metrics.

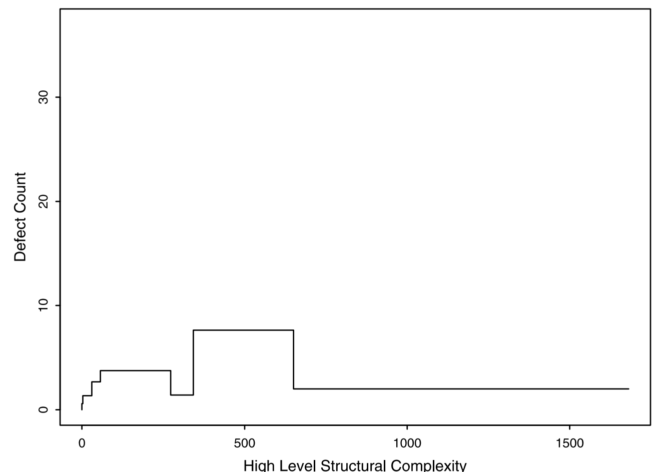


Fig. 3. Type B1 piece-wise linear model for IBM-LS.

Table 1
Metrics resulted in Type A and Type B piece-wise linear models

Product Name	Type A	Type B
IBM-LS	–	all 15 IBM-LS metrics
IBM-NS	Proc, HLSC, HLC	all other 8 IBM-NS metrics
NT-1	ArcNbr, CalUnqNbr, ComLogNbr, ComStrVol, CtrBrcWgt, CtrNstMax, CtrVol, CycCpl, HalVol, LocNbr, LopNbr, NdsNbr, NdsPndNbr, StmCtlNbr, VarGlbUsdNbr, VarUsdNbr.	all other 33 NT-1 metrics
NT-2	CalNbr, CalUnqNbr, CndCplMax, CtrNstMax, LocNbr, NdsEntNbr, CplLvl, CycCpl.	all other 41 NT-2 metrics
NT-3	CndNbr, ComStrVol, HalVol, LocNbr, KntNbr, NdsNbr, VarSpnSum	all other 42 NT-3 metrics
NT-4	CtrNstMax, LopNbr, PthIndNbr, CycCpl, StmCplAvg	all other 44 NT-4 metrics

We further divide the Type B category into two parts:

- Type B1: HD cluster is the one immediately preceding the HC cluster, such as in Fig. 3.
- Type B2: Type B but not Type B1, such as in Fig. 1a.

This further division of Type B models into B1 and B2 gives us additional information about the complexity ranking of the HD cluster. This issue will be examined further in Section 5.

The summary of our model type classification is given in Table 2. We obtained 82.43% Type B, with 29.28% Type B1 and 53.15% Type B2, and 17.57% Type A models. These can be interpreted as the preliminary results indicating that HC and HD clusters are usually different, or from different populations or distributions.

3.4. Data sets for hypothesis testing

To generalize the results, for any given product, we can accumulate the HC and HD clusters over the different metrics used, m's, into two module sets HC and HD as

$$\mathrm{HC} = \bigcup_m \mathrm{HC}_m$$

and

$$\mathrm{HD} = \bigcup_m \mathrm{HD}_m.$$

Then the same generic hypothesis testing can be carried out. Among the multiple observations corresponding to different cluster analyses for individual metric-defect count pairs, there are many duplicate entries representing the same modules. The union operations above remove those duplicates.

Other candidate data sets for our hypothesis testing include independently identified data sets Top$D$ and Top$C$ explained in Section 3.1. However, since Top$C$ selects largely the same modules and data points as HC, using fixed numbers or percentages of the highest complexity data points instead of variable size clusters, the conclusions drawn would be nearly identical. As a result, we did not include Top$C$ in our hypothesis testing. For Top$D$, we use Top$25D$ and Top$100D$, the top 25 and top 100 modules of a product respectively in the defect ranking. Both sizes are appropriate for making meaningful statistical inferences using the hypothesis testing technique explained in Section 4.1. We preferred having different sizes to confirm and compare our results.

To summarize, for any given product, our hypothesis testing will be carried out among the following pairs:

- HC vs. Top$25D$,
- HC vs. Top$100D$,
- HC vs. HD.

The numbers of data points in our samples are given in Table 3. Top$25D$ and Top$100D$ always include 25 and 100 data points respectively for any product.

Table 2
Number of metrics falling into Type A, Type B1, and Type B2 categories

Product name	Type A	Type B1	Type B2
IBM-LS	0	5	10
IBM-NS	3	0	8
NT-1	16	6	27
NT-2	8	11	30
NT-3	7	18	24
NT-4	5	25	19
Total	39 (17.57%)	65 (29.28%)	118 (53.15%)

4. Hypothesis testing

The motivation behind hypothesis testing is to provide statistical support to our preliminary results given in Section 3.3. We treat defect count as the random

Table 3
Numbers of data points in HC and HD for each product

	IBM-LS	IBM-NS	NT-1	NT-2	NT-3	NT-4
HC	257	611	620	857	656	878
HD	139	43	501	483	228	117

variable, and set forward several null hypotheses and corresponding alternative hypotheses concerning the distribution of this random variable in sample pairs. These null hypotheses and alternative hypotheses have a single generic form. A generic null hypothesis H_0 and its alternative hypothesis H_A concerning two samples, S_1 and S_2, can be written as:

H_0: Two samples have been drawn from the same population.
H_A: The defect distribution for population S_1 is shifted to the left or to the right of that for S_2.

Using statistical techniques, we test the null hypotheses and either accept them or reject them safely in favor of their alternative hypotheses.

4.1. Selecting appropriate statistical tests

We first examined the distribution of our samples to choose among the available statistical tests. We applied χ^2 goodness-of-fit test to check their normality. S-PLUS is used at this step to obtain the p-values, which were zero for all of the samples. These results showed that none of the samples have a normal distribution and also indicated that non-parametric methods are required in hypothesis testing.

Because of its applicability on the non-normal samples we have, we chose to apply a non-parametric, rank-sum method known as Mann–Whitney U test or Wilcoxon rank-sum test (Hamburg, 1977) supported by S-PLUS. Mann–Whitney U test shows whether two samples have been drawn from the same population, or equivalently from two different populations having the same mean. It is resistant to non-normal samples.

The assumptions of Mann–Whitney U test require that any two samples, S_1 and S_2, should:

1. be random samples independent from each other,
2. have a similar distribution shape and spread.

As stated in Section 3.1, Top$D$ is sampled independently of HC. Related to HC and HD sample pairs, although, according to any metric m used on any product, it is possible to have $HC_m = HD_m$ when it is a Type A model (see Section 3.3), it is not possible to say the tested samples, HC and HD, will be dependent. When we look at these samples, there is not an exact probability value p, which determines the data points in one sample according to the data points in the other one.

Looking at the histogram and box plots of our samples, we observed that the second assumption is also satisfied. The sample pairs had similar distribution shapes, that can be observed from the histogram plots. The box plots revealed that the variations of values in the sample pairs were also similar. An example can be seen in Fig. 4, where the histogram and box plots of HC and HD for IBM-NS are given.

4.2. Mann–Whitney U test for hypothesis testing

Mann–Whitney U test merges two samples, S_1 and S_2, which have n_1 and n_2 points respectively and sorts them

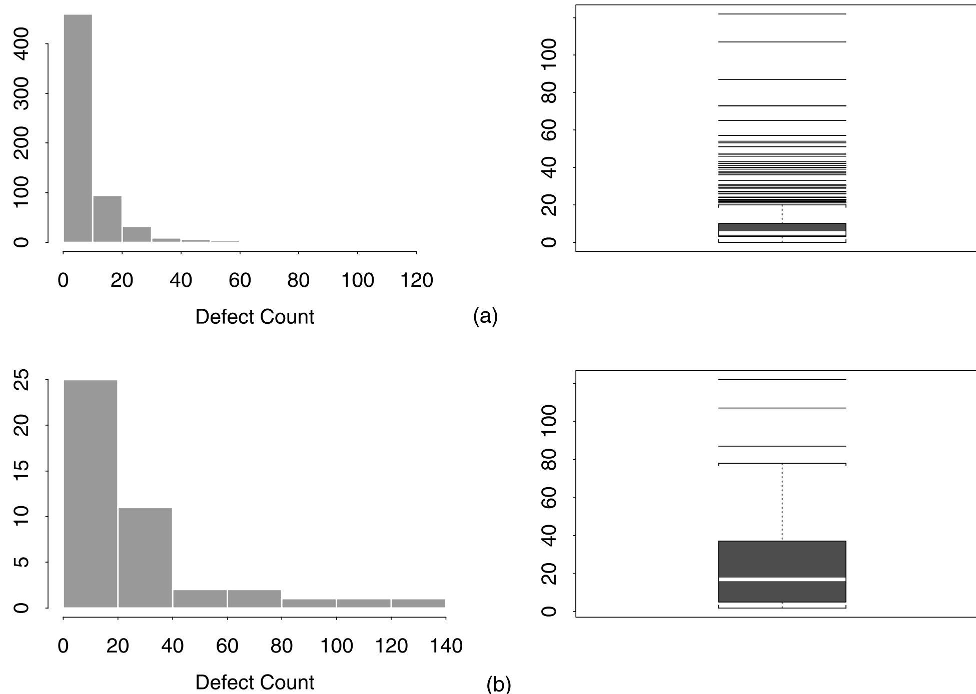


Fig. 4. Defect count histograms and box plots from IBM-NS: (a) HC and (b) HD.

to obtain a ranking sequence. Then the sum of the ranks, R_1 and R_2 are calculated for both samples. The test statistic U is obtained as below:

$$U = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

It is also possible to use R_2 with $n_2(n_2+1)$, which only changes the sign of U. U is a measure of the difference between two samples and it has a normal distribution. Its mean, μ_U, standard deviation, σ_U, and standard score, z, can be obtained as below:

$$\mu_U = \frac{n_1 n_2}{2}$$

$$\sigma_U = \sqrt{\frac{n_1 n_2 (n_1 + n_2 + 1)}{12}}$$

$$z = \frac{U - \mu_U}{\sigma_U}$$

Using a two-tailed Mann–Whitney U test, we can test our hypotheses by following the procedure below:

1. Produce the two samples, S_1 and S_2, subject to hypothesis testing.
2. Apply the Mann–Whitney U test statistic on the samples and obtain the z value above.
3. Decide upon a significance level, α. Then the critical value $|z_{\text{critical}}|$, is determined. α values of 0.05 and 0.01 correspond to 1.96 and 2.58 values of $|z_{\text{critical}}|$.
4. Accept H_0 if $|z| \leqslant |z_{\text{critical}}|$, otherwise reject H_0 in favor of H_A.

4.3. Test results

Each null hypothesis we tested is indicated as a row in Table 4 with the product subject to the analysis, the samples used (see Section 3.4) and the result obtained. The resulting z values of our hypothesis tests can be seen in Table 4, which range from −2.7851 to −15.0384. Both $\alpha = 0.05$ and $\alpha = 0.01$ allow us to reject the null hypotheses given in Table 4 according to step 4 of Section 4.2 because all these $|z|$ values exceeds $|z_{\text{critical}}|$. Consequently, all the null hypotheses for all sample pairs are rejected in favor of their corresponding alternative hypotheses. These results show that the HC and HD clusters we tested are not the same in defect distribution.

Table 4
Tested null hypotheses with their samples and resulting z values for different products

Product name	Samples		Results z
	S_1	S_2	
IBM-LS	HC	Top 25D	−7.9312
	HC	Top 100D	−11.9387
	HC	HD	−6.3903
IBM-NS	HC	Top 25D	−8.2194
	HC	Top 100D	−14.1263
	HC	HD	−4.9644
NT-1	HC	Top 25D	−8.3677
	HC	Top 100D	−14.0605
	HC	HD	−2.7851
NT-2	HC	Top 25D	−8.3571
	HC	Top 100D	−14.7316
	HC	HD	−7.0602
NT-3	HC	Top 25D	−8.3441
	HC	Top 100D	−13.9564
	HC	HD	−9.6969
NT-4	HC	Top 25D	−8.5951
	HC	Top 100D	−15.0384
	HC	HD	−8.8462

5. Complexity ranking of top defect clusters

After obtaining the results indicating that HC clusters are statistically identical with neither HD clusters nor top defect clusters, we aimed to provide more constructive information by placing the top defect clusters in the complexity ranking. Such information can help us better understand and characterize these modules for focused quality assurance activities. Since the number of modules is different in each product, we chose to analyze the top 2% (Top 2%D) and top 5% (Top 5%D) of the modules according to defect ranking, instead of analyzing a fixed number of modules as the top defect cluster, to make this characterization easier to perform.

The complexity rankings of Top 2%D and Top 5%D modules can be calculated from the original complexity values using S-PLUS (Venables and Ripley, 1994). The ranks of the same values are assigned as the average of their ranks. Table 5 summarizes our results for each product by giving cluster names and sizes with the summary data of their corresponding rank sets.

In Table 5, the maximum rank column shows that very complex, sometimes even the most complex modules are in the top defect clusters. The possible rank range, shown as a separate column, from the least complex to the most complex is between 1 and the number of modules in the related product. However, looking at the minimum rank column, it can be seen that very low complexity, or sometimes even the least complex modules are also involved in these clusters. Therefore, we also calculated average rank and average rank percentile values.

The clusters we examined occupy fairly high places in the complexity ranking but not the highest. If they had the highest complexity, Top 2%D and Top 5%D would correspond to the average complexity rank percentiles close to 99% and 97.5% respectively. The averages of the complexity percentile values given in the last column of

Table 5
Complexity ranking of top defect clusters

Product name	Cluster name	Cluster size	Min. rank	Max. rank	Possible rank range	Avg. rank	Avg. rank percentile (%)
IBM-LS	Top 2%*D*	26	413.5	1295	1–1302	1107.4	85.51
	Top 5%*D*	65	128.5	1295	1–1302	1081.1	83.48
IBM-NS	Top 2%*D*	20	53	995	1–995	800	80.40
	Top 5%*D*	50	53	995	1–995	786	78.99
NT-1	Top 2%*D*	16	2	804	1–804	564.5	70.21
	Top 5%*D*	40	2	804	1–804	552.8	68.76
NT-2	Top 2%*D*	22	2	1098	1–1098	786.7	71.65
	Top 5%*D*	55	2	1098	1–1098	735.3	66.97
NT-3	Top 2%*D*	14	2	711	1–712	494.5	69.55
	Top 5%*D*	36	1	711	1–712	468.8	65.94
NT-4	Top 2%*D*	18	1	900	1–900	730.6	81.18
	Top 5%*D*	45	1	900	1–900	685.2	76.13

Table 5 range from 69.55% to 85.51% (with an average of 76.42%) for Top 2%*D* and from 65.94% to 83.48% (with an average of 73.38%) for Top 5%*D* modules. These numbers give us an idea about the location of the clusters we examined in the complexity ranking. The lowest quality modules are not always the most complex modules but they usually are the ones slightly below the most complex modules in the complexity ranking.

6. Discussions, conclusions and perspectives

In this paper, we analyzed multiple sets of product data from two different industries, covering millions of lines of code and thousands of software modules. The comparison of the sets of HD modules and the sets of HC modules through hypothesis testing has conclusively rejected our null hypothesis, in favor of the alternative hypothesis that these two kinds of modules are statistically different. Our result has several important implications:

- Although defect count and measured complexity are generally positively correlated, as confirmed by our previous studies (Tian et al., 2001; Tian and Troster, 1998) for the same products we studied in this paper, monotonic models based on correlations on the raw or (monotonically) transformed data can not be used effectively to identify HD modules under many situations, because it would imply our null hypothesis to be true.
- Much of the above point was recognized by many researchers, who focused on developing and using alternative risk identification techniques based on pattern matching, tree-based models, neural networks, etc. in analyzing software engineering data and guiding remedial actions focused on the identified high risk areas (Briand et al., 1993; Khoshgoftaar et al., 1997; Selby and Porter, 1988). This research provides additional empirical evidence to support further research and development in this research direction.

Furthermore, unlike similar studies such as (Fenton and Ohlsson, 2000), which pointed out the inability to explain observed faults by size and complexity, we have also characterized the complexity behavior of HD modules. This characterization provides more constructive information that can help us focus on modules demonstrating similar complexity behavior. In particular, we observe that the HD modules are typically those measured at fairly high percentile on various complexity scales, but not the highest. This empirical characterization also has various interesting implications:

- This phenomenon has been observed by many software practitioners, including many of our colleagues and collaborators at IBM and Nortel Networks. One conjecture often mentioned by these practitioners is that many complex modules are intrinsically complex because of the problem they are dealing with, and recognized as such. Consequently, highly skilled personnel and adequate effort were allocated to such modules, resulting in their relative high quality or low defect counts. Similar conjectures have also been empirically validated in Emam et al. (2001), where after controlling for size, none of the metrics were associated with fault-proneness anymore. Our results generally support such conclusions.
- Our study also points to the possibility of some *worst* complexity, those "not too big (complex) not too small (simple)", which might contain the highest number of defects. This non-monotonic phenomenon is similar to the "Goldilocks Conjecture" (the possibility of an optimal size for quality) mentioned in Fenton and Neil (1999), but in the opposite direction. Our results indicate that we should pay particular at-

tention to modules whose measured complexity falls slightly below the most complex ones, and point out the importance of searching for alternative techniques to identify these HD modules for effective quality assurance and improvement.

As an immediate follow-up to our study, we are seeking opportunities to analyze additional data sets, preferably from different industries, to further validate our results and make them more generalizable to a wider variety of product types and industries. Such additional effort will help us better understand the measurement characteristics and behavior of HD modules, search for more effective techniques to identify such modules, and work toward the goal of an optimal strategy that maximizes quality improvement under project constraints for many industrial applications.

Acknowledgements

This research is supported in part by NSF/CAREER Award CCR-9733588, THECB/ATP Awards 003613-0030-1999 and 003613-0030-2001, IBM, and Nortel Networks. We would like to thank Dave Frame, Tony Nguyen, and others at Nortel Networks for their help throughout this work. We also thank the anonymous reviewers for their constructive comments and suggestions.

Appendix A. Metrics for IBM products

The names of the metrics used for the analyzed IBM products are given in Table 6. All of the 15 metrics in Table 6 were used in IBM-LS, whereas 11 of them excluding Rdecl, Udecl, CSI, SMI were used in IBM-NS. Most of the metrics given in Table 6 are commonly used complexity metrics described in Fenton and Pfleeger (1996). We give a classification of these metrics below:

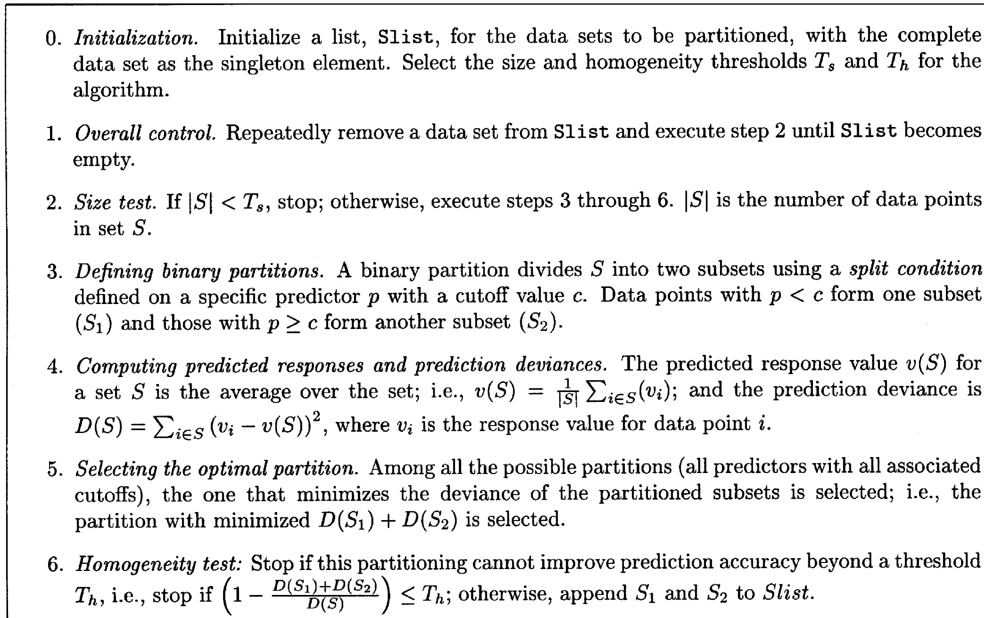
- Code complexity metrics: McCabe, Proc, Stmt, RDecl, UDecl.
- Design complexity metrics: The design complexity metrics include high level and module level metrics. High level metrics (HLSC, HLDC, and HLC) measure inter-module coupling and data flow. Module

Table 6
Metrics for IBM products

Abbreviation	Metric name
McCabe	Number of independent execution paths
Proc	Number of internal procedures
Stmt	Number of executable statements
Rdecl	Number of referenced declared variables
Udecl	Number of unused declared variables
HLSC	High-level structural complexity
HLDC	High-level data complexity
HLC	High-level system complexity
MLSC	Module-level structural complexity
MLDC	Module-level data complexity
MLC	Module-level system complexity
LOC	Lines of code
Com	Lines of comments
CSI	Changed lines of code
SMI	Software maturity index

Table 7
Metrics for Nortel Networks products

Abbreviation	Metric name
ArcComAvg	Commented arcs percentage
ArcNbr	Number of arcs
ArcWgt	Weight of arcs
CalNbr	Number of calls to others
CalUnqNbr	Unique calls to others
CndCplAvg	Average conditional arc complexity
CndCplMax	Maximal conditional arc complexity
CndNbr	Number of conditional arcs
CndSpnAvg	Average conditional arc span
CndSpnMax	Maximum conditional arc span
ComDecVol	Number of alphanumeric characters in the comments in the declarations sessions
ComLogNbr	Number of logical comments
ComStrAvg	Comments volume ratio
ComStrVol	Comments volume in structure
CtrBrcWgt	Weighted number of branches in control structure
CtrComAvg	Average commented control structure
CtrNstAvg	Average control structure nesting
CtrNstMax	Maximal control structure nesting
CtrNstWgt	Weighted mean control structure nesting
CtrVol	Control flow structural volume
HalDif	Halstead program difficulty
HalEff	Halstead program effort
HalLen	Halstead program length
HalLvl	Halstead program level
HalVoc	Halstead program vocabulary
HalVol	Halstead program volume
KntNbr	Number of knots
LocNbr	Number of lines of code
LopNbr	Number of loop constructs
LopStrAvg	Average loop structure
LopStrWgt	Weighted loop structure
NdsEntNbr	Number of entry nodes
NdsExtNbr	Number of exit nodes
NdsNbr	Number of nodes
NdsPndNbr	Number of pending nodes
PthIndNbr	Number of independent paths
CplLvl	Routine complexity level
CycCpl	Cyclomatic number
StmCplAvg	Average complexity of statements
StmCtlNbr	Number of control statements
StmDecNbr	Number of declarative statements
StmExeNbr	Number of executable statements
StmNbr	Number of statements
VarGlbUsdNbr	Number of occurrences of global variable usage
VarLenAvg	Average length of variable names
VarSpnMax	Maximal variable span
VarSpnSum	Sum of all variables span
VarUsdNbr	Number of variables used
VarUsdUnqNb	Number of distinct variables used



0. *Initialization.* Initialize a list, `Slist`, for the data sets to be partitioned, with the complete data set as the singleton element. Select the size and homogeneity thresholds T_s and T_h for the algorithm.
1. *Overall control.* Repeatedly remove a data set from `Slist` and execute step 2 until `Slist` becomes empty.
2. *Size test.* If $|S| < T_s$, stop; otherwise, execute steps 3 through 6. $|S|$ is the number of data points in set S.
3. *Defining binary partitions.* A binary partition divides S into two subsets using a *split condition* defined on a specific predictor p with a cutoff value c. Data points with $p < c$ form one subset (S_1) and those with $p \geq c$ form another subset (S_2).
4. *Computing predicted responses and prediction deviances.* The predicted response value $v(S)$ for a set S is the average over the set; i.e., $v(S) = \frac{1}{|S|}\sum_{i \in S}(v_i)$; and the prediction deviance is $D(S) = \sum_{i \in S}(v_i - v(S))^2$, where v_i is the response value for data point i.
5. *Selecting the optimal partition.* Among all the possible partitions (all predictors with all associated cutoffs), the one that minimizes the deviance of the partitioned subsets is selected; i.e., the partition with minimized $D(S_1) + D(S_2)$ is selected.
6. *Homogeneity test:* Stop if this partitioning cannot improve prediction accuracy beyond a threshold T_h, i.e., stop if $\left(1 - \frac{D(S_1)+D(S_2)}{D(S)}\right) \leq T_h$; otherwise, append S_1 and S_2 to *Slist*.

Fig. 5. Algorithm for tree-based model construction.

level metrics (MLSC, MLDC, and MLC) measure control flow and data flow among the procedures in a module. Both high-level and module-level metrics are calculated using fanout, and the number of I/O variables. More information can be found in Card and Glass (1990).

- Size and change metrics: LOC, Com, CSI, SMI (ratio of unchanged lines of code divided by LOC).

Appendix B. Metrics for Nortel Networks products

The names of the 49 metrics used for the Nortel Networks products are given in Table 7. These commonly known complexity metrics, most of which are also described in Fenton and Pfleeger (1996), can be classified according to Hudepohl et al. (1996b). For each class, all the metrics of Table 7 that fall into that class are listed below:

- Volume: NdsEntNbr, NdsExtNbr, LocNbr, StmNbr, StmDecNbr, StmExeNbr, StmCtlNbr, ArcNbr, NdsNbr, CycCpl, CalNbr, CalUnqNbr, ComLogNbr, ComDecVol, ComStrVol, CtrVol.
- Testability: CtrNstWgt, CtrNstMax, CtrNstAvg, CndSpnMax, CndSpnAvg, LopNbr, LopStrAvg, LopStrWgt.
- Decision complexity: CndCplAvg, CndCplMax.
- Independent paths: PthIndNbr.
- Structuredness: KntNbr.

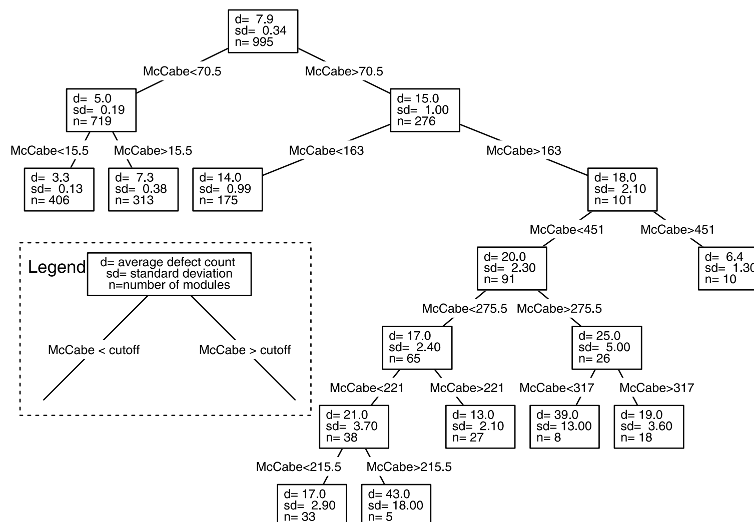


Fig. 6. A sample tree-based model for IBM-NS.

- Dead code: NdsPndNbr.
- Readability: VarLenAvg, CtrComAvg, ArcComAvg, ComStrAvg.

Appendix C. Tree-based modeling and a sample model

Tree-based modeling is a statistical analysis technique that attempts to establish predictive relations through recursive partitioning (Clark and Pregibon, 1993). In tree-based models, modeling results are represented in tree structures. Each node in a tree represents a set of data, which is recursively partitioned into smaller subsets. The data used in such models consist of multiple attributes, with one attribute identified as the *response* variable and several other attributes identified as *predictor* variables. Recursive partitioning minimizes the difference between predicted response values and the observed response values. The specific tree construction algorithm supported by S-PLUS and used in this paper to identify HC and HD clusters is summarized in Fig. 5. Fig. 6 depicts a sample model by treating defect count as the response variable and McCabe's cyclomatic complexity as the predictor variable.

References

Briand, L.C., Basili, V.R., Hetmanski, C.J., 1993. Developing interpretable models with optimal set reduction for identifying high-risk software components. IEEE Trans. Software Eng. 19 (11), 1028–1044.

Card, D.N., Glass, R.L., 1990. Measuring Software Design Quality. Prentice Hall, Englewood Cliffs, NJ.

Clark, L.A., Pregibon, D., 1993. Tree based models. In: Chambers, J.M., Hastie, T.J. (Eds.), Statistical Models in S. Chapman and Hall, London, pp. 377–419 (Chapter 9).

Emam, K.E., Benlarbi, S., Goel, N., Rai, S.N., 2001. The confounding effect of class size on the validity of object-oriented metrics. IEEE Trans. Software Eng. 27 (7), 630–650.

Fenton, N., Neil, M., 1999. A critique of software defect prediction models. IEEE Trans. Software Eng. 25 (5), 675–689.

Fenton, N., Pfleeger, S.L., 1996. Software Metrics: A Rigorous and Practical Approach, second ed., PWS Publishing.

Fenton, N.E., Ohlsson, N., 2000. Quantitative analysis of faults and failures in a complex software system. IEEE Trans. Software Eng. 26 (8), 797–814.

Halstead, M.H., 1977. Elements of Software Science. Elsevier.

Hamburg, M., 1977. Statistical Analysis for Decision Making, second ed., Harcourt Brace Jovanovich, New York, NY.

Hudepohl, J., Aud, S., Khoshgoftaar, T., Allen, E., Mayrand, J., 1996a. Integrating metrics and models for software risk assessment. In: Proceedings of the 7th International Symposium on Software Reliability Engineering, pp. 93–98.

Hudepohl, J.P., Aud, S.J., Khoshgoftaar, T.M., Allen, E.B., Mayrand, J., 1996b. Emerald: Software metrics and models on the desktop. IEEE Software 13 (5), 56–60.

Khoshgoftaar, T.M., Allen, E.B., Hudepohl, J., Aud, S., 1997. Applications of neural networks to software quality modeling of a very large telecommunications system. IEEE Trans. Neural Networks 8 (4), 902–909.

McCabe, T.J., 1976. A complexity measure. IEEE Trans. Software Eng. 2 (6), 308–320.

Munson, J.C., Khoshgoftaar, T.M., 1992. The detection of fault-prone programs. IEEE Trans. Software Eng. 18 (5), 423–433.

Porter, A.A., Selby, R.W., 1990. Empirically guided software development using metric-based classification trees. IEEE Software 7 (2), 46–54.

Selby, R.W., Porter, A.A., 1988. Learning from examples: Generation and evaluation of decision trees for software resource analysis. IEEE Trans. Software Eng. 14 (12), 1743–1757.

Tian, J., 1995. Integrating time domain and input domain analyses of software reliability using tree-based models. IEEE Trans. Software Eng. 21 (12), 945–958.

Tian, J., Nguyen, A., Allen, C., Appan, R., 2001. Experience with identifying and characterizing problem prone modules in telecommunication software systems. J. Syst. Software 57 (3), 207–215.

Tian, J., Troster, J., 1998. A comparison of measurement and defect characteristics of new and legacy software systems. J. Syst. Software 44 (2), 135–146.

Venables, W.N., Ripley, B.D., 1994. Modern Applied Statistics with S-Plus. Springer-Verlag, New York.

A. Güneş Koru received a B.S. in Computer Engineering degree from Ege University, Izmir, Turkey in 1996 and an M.S. in Computer Engineering degree from Dokuz Eylul University, Izmir, Turkey in 1998. Currently, he is in the Ph.D. in Computer Science program at Southern Methodist University, Dallas, TX, where he also received a M.S. degree in Software Engineering. His research interests include software testing, measurement, reliability, and architecture. He is a member of IEEE and ACM.

Jeff (Jianhui) Tian received a B.S. degree in Electrical Engineering from Xi'an Jiaotong University in 1982, a M.S. degree in Engineering Science from Harvard University in 1986, and a Ph.D. degree in Computer Science from the University of Maryland in 1992. He worked for the IBM Software Solutions Toronto Laboratory between 1992 and 1995 as a software quality and process analyst. Since 1995, he has been with Southern Methodist University, Dallas, Texas, now as an Associate Professor of Computer Science and Engineering, with joint appointment at the Dept. of Engineering Management, Information and Systems. His current research interests include software testing, measurement, reliability, safety, complexity, and telecommunication software and systems. He is a member of IEEE and ACM.