Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement
Jeff Tian

CHAPTER 21

RISK IDENTIFICATION FOR QUANTIFIABLE QUALITY IMPROVEMENT

This chapter describes and compares risk identification techniques that can be used to identify high-risk (low-quality) areas for focused quality improvement. Each technique is briefly described and illustrated with practical application examples from industrial or governmental projects. The techniques are compared using several criteria, including simplicity, accuracy and stability of results, ease of result interpretation, and utility in guiding quality assurance and improvement. A comprehensive example using a specific risk identification technique to analyze defect data classified according to ODC (Chapter 20) is also included.

21.1 BASIC IDEAS AND CONCEPTS

As described in Chapter 2, a *defect* generally refers to a problem in the software, which may lead to undesirable consequences for both the software development/maintenance organizations and the software users. The potential for such undesirable consequences, including schedule delays, cost overruns, and highly defective software products, is usually referred to as *risk*. Various statistical analyses and learning algorithm based techniques have been developed or adopted to identify and reduce such risks.

On the other hand, fault distribution is highly uneven for most software products, regardless of their size, functionality, implementation language, and other characteristics. Much empirical evidence has accumulated over the years to support the so-called 80:20 rule, which states that about 20% of the software components are responsible for about 80% of the problems (Porter and Selby, 1990; Tian and Troster, 1998; Boehm and Basili, 2001),

as also demonstrated by various defect distribution and analysis examples in the previous chapter. These problematic components represent high risks to both the software development/maintenance organizations and software users. Therefore, there is a great need for risk identification techniques to analyze these measurement data so that inspection, testing, and other quality assurance activities can be more effectively focused on those potentially high-defect components.

Similar concepts about risk related to other entities of concern, such as schedule, cost, or reliability related risk can also be defined (Boehm, 1991). As we will describe in Chapter 22, reliability problems and related risk can be addressed through the use of appropriate risk identification techniques to identify low-reliability areas for focused reliability improvement.

To measure and characterize these high-risk or potentially high-defect modules, various software metrics can be used to capture information about software design, code, size, change history, etc. (Fenton and Pfleeger, 1996), as well as other product or process characteristics we described in Chapter 18. Once the measurement data are collected from existing project databases or calculated using measurement tools, various techniques can then be employed to analyze the data in order to identify high-risk modules. The basic idea of risk identification is to use predictive modeling to focus on the high-risk areas, as follows:

- First, we need to establish a predictive relationship between project metrics and actual product defects based on historical data.
- Then, this established predictive relation is used to predict potential defects for the new project or new product release once the project metrics data become available, but before actual defects are observed in the new project or product release.
- In the above prediction, the focus is on the high-risk or the potentially high-defect modules or components.

Following the discussion in Chapter 20, we primarily use DF, or defect fixes as our defect measurement in this chapter, and relate it to other project measurements through measurement-based predictive models covered in Chapter 19, particularly those tailored for risk identification and analyses.

Like any other statistical technique, these risk identification techniques cannot establish proof of a causal relationship. However, they can provide some strong evidence that there may be a causal relationship in an observed effect. By extracting the specific characteristics of existing high-defect modules, these analyses can help software professionals identify new modules demonstrating similar measurement characteristics and take early actions to reduce risks or prevent potential problems. Appropriate risk identification techniques can be selected to fit specific application environments in order to identify high-risk software components for focused inspection and testing.

A preliminary survey of these risk identification techniques and their comparison can be found in Tian (2000), including: traditional statistical analysis techniques, principal component analysis and discriminant analysis, neural networks, tree-based modeling, pattern matching techniques, and learning algorithms. In this chapter, these techniques are described and illustrated with practical examples from industrial and governmental projects. Data, models, and analysis results presented in this chapter are extracted from several commercial software products from IBM (Tian, 1995; Khoshgoftaar and Szabo, 1996; Tian and Troster, 1998), governmental projects from NASA (Porter and Selby, 1990; Briand et al.,

1993), as well as software systems used in aerospace, medical, and telecommunication industries (Munson and Khoshgoftaar, 1992; Khoshgoftaar et al., 1996; Tian et al., 2001).

In addition, we compare these risk identification techniques according to several criteria, including: accuracy, simplicity, early availability and stability, ease of result interpretation, constructive information and guidance for quality improvement, and availability of tool support. We conclude the chapter with our recommendation for an integrated life-cycle approach where selected techniques can be used effectively through software development for quality assurance and improvement.

21.2 TRADITIONAL STATISTICAL ANALYSIS TECHNIQUES

Various traditional statistical analysis techniques (Venables and Ripley, 1994) can be used to understand the general relations between defects and various other software measurement data. These statistical relations and the general understanding can be used to a limited degree to identify high-defect modules.

Correlation analysis

The statistical correlation between two random variables x and y can be captured by the (linear) correlation coefficient $c(x, y)$, which ranges between -1 and 1. A positive correlation indicates that the two variables are generally moving in the same direction (for example, a larger x is usually accompanied by a larger y); while a negative correlation indicates the opposite. The closer to 1 the absolute value $|c(x, y)|$ is, the more tightly correlated x and y are. Because software measurement data are often skewed, such as in Table 20.3, where many modules contain few DF while a few modules contain many, rank correlations are often calculated in addition to the linear correlations.

If the observed defects are highly correlated to a software metric, we can then identify those modules with larger (or smaller, if negatively correlated) values of the given metric as high-defect modules. However, DF-metric correlations are generally low (Card and Glass, 1990; Fenton and Pfleeger, 1996), which limits our ability to predict high-defect modules based on metrics data. For example, the highest DF-metric correlation is 0.731, between DF and CSI(changed source instructions or changed lines of code), for the product LS (Tian and Troster, 1998).

Linear regression models

Linear regression models express a selected random variable y, referred to as the dependent variable, as a linear combination of n other random variables, $x_1, x_2, \ldots, x_n$, referred to as independent variables, in the form of:

$$y = \alpha_0 + \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n + \epsilon,$$

where ϵ is the error term, and parameters $\alpha_0, \alpha_1, \ldots, \alpha_n$ can be estimated from the observation data. Because of the data skew, logarithmic transformation of data can also be used, yielding a log-linear regression.

When regression models are fitted to defect and metrics data, DF can be expressed as a linear or log-linear function of other metrics. The correlation coefficient between the observed defects and the fitted linear or log-linear models, or the square root of the

corresponding multiple R-squared value, can be interpreted in similar ways as for DF-metric correlations. However, these corrections are 0.767 and 0.789 respectively for the product LS (Tian and Troster, 1998), — only slightly higher than correlations between DF and the individual metrics. Similar patterns were also observed in studies of other products. In general, linear or log-linear regression models suffer from similar shortcomings as correlation analysis models, and do not perform well in predicting high-defect modules. In addition, parameter estimates for these models are usually unstable due to high correlation in the metrics data.

Other models and general observations

Various other traditional statistical analysis techniques, such as non-linear regression models, generalized additive models, logistic regression models, etc. (Venables and Ripley, 1994), can also be used to identify high-defect modules. Another alternative to use traditional statistical models on these skewed measurement data is to perform data transformation before modeling. For example, logarithmic transformation is a commonly used technique to deal with data skewed towards the lower end (those with just a few data points at the higher end). But to overcome the undefined values for $\log 0$, which may even be the majority of data point, such as the 58.8% modules with 0 defects in Table 20.3, alternative transformations such as $y = \log(x + 1)$ to transform original data x to transformed data y. However, such transformations obscure the relationship between different entities and make result interpretation harder.

In general, all the above models suffer from similar limitations as correlation and linear regression models described earlier. The key problem is the data treatment: data from the majority of low-defect modules dominate these statistical results, which contain little information about high-defect modules. To overcome these limitations, alternative analysis techniques need to be used, as described below.

21.3 NEW TECHNIQUES FOR RISK IDENTIFICATION

Recently, various new techniques have been developed or adapted for risk identification purposes, including classification and analysis techniques based on statistical analysis, learning, and pattern matching. We next describe these techniques and illustrate how they can be used to identify high-defect modules.

21.3.1 Principal component and discriminant analyses

Principal component analysis and discriminant analysis are useful statistical techniques for multivariate data (Venables and Ripley, 1994). The former reduces multivariate data into a few orthogonal dimensions; while the latter classifies these data points into several mutually exclusive groups. These analysis techniques are especially useful when there are a large number of correlated variables in the collected data. Software metrics data fit into this scenario, where many closely related metrics exist to measure design, size, and complexity of the data and control structures in the code (therefore, the measurement results are correlated, too).

The principal components are formed by linear combinations of the original data variables to form an orthogonal set of variables that are statistically uncorrelated. If the original data with n variables are linearly independent (that is, none of the variables can be expressed

Table 21.1 Principal components for a commercial product

	pc1	pc2	pc3	pc4
eigenvalue λ_i	2.352	1.296	1.042	0.711
% of variance	55.3%	16.8%	10.8%	5.1%
cumulative % of variance	55.3%	72.1%	82.9%	88.0%

as linear combinations of other variables), then their covariance matrix, Σ, an $n \times n$ matrix, can be expressed as its eigendecomposition,

$$\Sigma = C^T \Lambda C,$$

where Λ is a diagonal matrix with eigenvalues λ_i, $i = 1, 2, \ldots, n$, in decreasing order (representing decreasing importance). The original measurement data matrix Z can be transformed into the corresponding principal-component data matrix D using a transformation function also defined by the eigenvalues.

Table 21.1 gives the first 4 principal components (pc1 $\sim$ pc4) for the product NS (Tian and Troster, 1998), where the original data contain 11 variables (for 11 different metrics for modules in NS). Among the principal components, pc1 $\sim$ pc4 explain 88% of the total variance. As a result, the original data can be reduced to these four principal components, without much loss of information.

Once a few important principal components are extracted, they can be used in various models to identify high-defect modules. For example, selected principal components were used with discriminant analysis to classify software modules into *fault-prone* and *other* ones for software systems used in aerospace, medical, and telecommunication industries (Munson and Khoshgoftaar, 1992; Khoshgoftaar et al., 1996). The models using principal components have several advantages over similar models using the original (raw) data: The models are simpler because fewer independent variables are used. The parameter estimates are also more stable due to the orthogonality among the principal components.

Discriminant analysis is a statistical analysis technique that classifies multivariate data points or entities, such as software modules characterized by different metrics, into mutually exclusive groups. This classification is done by using a discriminant function to assign data points to one of the groups while minimizing within group differences. For example, a discriminant function defined on selected principal components was derived to separate fault-prone software modules from the rest for some telecommunication software developed in Nortel (Khoshgoftaar et al., 1996):

- Assign d_i to G_1, if

$$\frac{f_1(d_i)}{f_2(d_i)} > \frac{\pi_2}{\pi_1},$$

  where the entities are defined as:

  - d_i is the i-th modules principal-component values (i-th row of the D matrix above).
  - G_1 and G_2 are mutually exclusive classes representing normal (not fault-prone) and fault-prone modules respectively.
  - π_k is the prior probability of membership in G_k.
  - $f_k(d_i)$ gives the probability that d_i is in G_k.

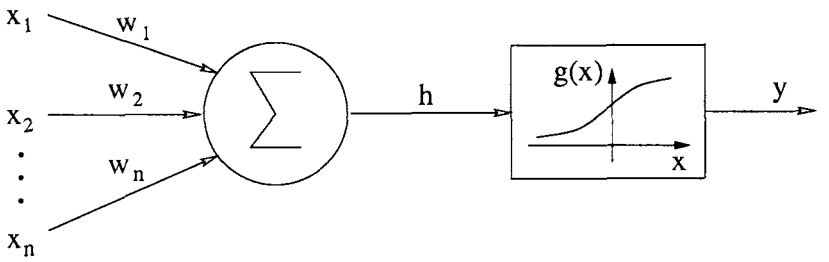

Figure 21.1 Processing model of a neuron

- Otherwise, assign d_i to G_2.

These applications have yielded fairly accurate results for grouping modules in the current project, with misclassification rate at about 1%, and for predictions into the future, with misclassification at 22.6% or 31.1% for the two models used in Khoshgoftaar et al. (1996)..

21.3.2 Artificial neural networks and learning algorithms

Artificial neural networks are based on learning algorithms inspired by biological neural networks, and can be used to solve various challenging problems, including pattern classification, categorization, approximation, etc. (Jain et al., 1996). Processing of an individual neuron is depicted by Figure 21.1, with:

$$h = \sum_{i=1}^{n} w_i x_i \quad \text{and} \quad y = g(h),$$

where x_1, x_2, ..., x_n are the input, w_1, w_2, ..., w_n the input weights, g the activation function, and y the output. The commonly used activation functions include threshold, piecewise linear, sigmoid, and Gaussian. Sigmoid function depicted in Figure 21.1 and used in Khoshgoftaar and Szabo (1996) is defined by:

$$g(x) = \frac{1}{1 + e^{-\beta x}}.$$

An artificial neural network is formed by connecting individual neurons in a specific network architecture.

When an artificial neural network is applied to a given data set, an iterative learning procedure can be followed to minimize the network error, or the difference between the predicted and actual output. This can be achieved by following various learning algorithms to adjust the weights at individual neurons. One of the most widely used such algorithms is backward propagation, summarized in Figure 21.2.

Recently, artificial neural network models were used to identify high-defect modules for some system software (Kernel.1, Kernel.2, and Kernel.3) developed in IBM (Khoshgoftaar and Szabo, 1996). Both the raw data and the principal component data from Kernel.1 were used as input to the models, starting with a small number (20) of hidden layer neurons and gradually adding more neurons until the model converge. 40 hidden layer neurons were needed for the model with raw data as input to converge; while only 24 were needed for the principal component data. In addition, as show in Table 21.2, once they were

0. *Initialization:* Initialize the weights to small random values.
1. *Overall control:* Repeat steps 2 $\sim$ 6 until the error in the output layer is below a pre-specified threshold or a maximum number of iterations is reached.
2. Randomly choose an input.
3. Propagate the signal forward through the network.
4. Compute the errors in the output layer.
5. Compute the deltas for the preceding layers by propagating the errors backward.
6. Update the weights based on these deltas.

Figure 21.2 Backward propagation algorithm for artificial neural networks

Table 21.2 Predicting defects using artificial neural networks

System	Model Data	Output Error			
		mean	std.dev	min.	max.
Kernel.2	raw	11.4	6.6	0.19	32.8
	principal components	7.1	5.6	0.05	42.8
Kernel.3	raw	11.0	6.3	0.12	31.6
	principal components	4.7	4.1	0.02	26.2

applied to Kernel.2 and Kernel.3, the model based on principal components outperformed the one based on raw data by a significant margin. The combination of principal component analysis and neural networks also outperformed linear regression models (Khoshgoftaar et al., 1996). This combination offers an effective and efficient (takes less time to train the model with relatively fewer neurons) alternative to identify high-defect modules for quality improvement.

21.3.3 Data partitions and tree-based modeling

In general, different modules of a large software system may possess quite different characteristics because of the diverse functionalities, program sources, and evolution paths. Sometimes, it is not the particular values but specific ranges that have practical significance. Arguably, such data are more properly handled if they are partitioned, and analyzed separately to accommodate for the qualitative differences among the partitioned subsets. In this way, high-defect modules with different characteristics for different partitions can be identified, and different actions can be carried out to correct the problems.

Tree-based modeling (Clark and Pregibon, 1993) is a statistical analysis technique that handles data partitions and related analysis. The model construction involves the data set being recursively partitioned, using split conditions defined on selected predictors (or independent variables), into smaller subsets with increasing homogeneity of response (or dependent variable) values. The binary partitioning algorithm, supported by the commercial software tool S-PLUS is summarized in Figure 21.3. Each subset of data associated with a

0. *Initialization.* Set the list, `Slist`, to contain only the complete data set as its 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. For numerical p, it can be defined 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). If p is a categorical variable, a binary partition is a unique grouping of all its category values into two mutually exclusive subsets S_1 and 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; that is, $v(S) = \frac{1}{|S|}\sum_{i \in S}(v_i)$; 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 or binary groupings), the one that minimizes the deviance of the partitioned subsets is selected; that is, the partition with minimized $D(S_1) + D(S_2)$ is selected.

6. *Homogeneity test:* Stop if $\left(1 - \frac{D(S_1)+D(S_2)}{D(S)}\right) \leq T_h$ (that is, stop if there is no substantial gain in prediction accuracy in further partitioning); otherwise, append S_1 and S_2 to $Slist$.

Figure 21.3 Algorithm for tree-based model construction

tree node is uniquely described by the path and associated split conditions from the root to it. The results presented in such forms are natural to the decision process and, consequently, are easy to interpret and easy to use. The characterization of individual nodes and associated data subsets can also help us understand subsets of high-defect modules, and therefore can be used to guide remedial actions focused on those identified modules and on modules demonstrating similar characteristics in related products.

Tree based models were first used in Porter and Selby (1990) to analyze data from NASA Software Engineering Laboratory, where various software metrics data were used to predict project effort and to identify high-risk areas for focused remedial actions. Recently, it was used to identify high-defect modules for several commercial software products (Tian and Troster, 1998). Figure 21.4 shows a tree-based model constructed for NS, one of these products, relating DF to 11 other design, size, and complexity metrics. The specific metrics selected by the tree construction algorithm include:

- HLSC, or high-level structural complexity (Card and Glass, 1990), a design complexity metric reflecting the number of external function calls.

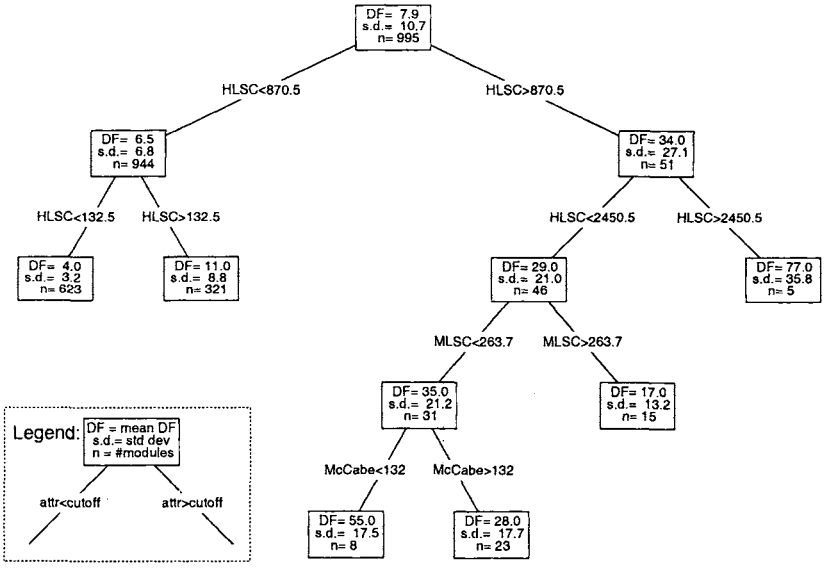

Figure 21.4 Tree-based defect model for a commercial product

Table 21.3 Characterizing high-defect modules for a commercial product

Node:	Split Conditions/Subset Characteristics	#Modules	Predicted-DF
rlll:	$870.5 < \texttt{HLSC} < 2450.5, \texttt{MLSC} < 263.7, \texttt{McCabe} < 132$	8	55.0
rr:	$\texttt{HLSC} > 2450.5$	5	77.0

- MLSC, or module-level structural complexity (Card and Glass, 1990), a design complexity metric reflecting the number of internal function calls within the module.
- McCabe, or McCabe's cyclomatic complexity (McCabe, 1976), a program complexity metric defined to be the number of independent control flow paths for a given program.

The subsets with extremely high DF can be easily identified as those associated with leaf nodes "rlll" and "rr" in Figure 21.4. Each node is labeled by the series of decisions, "l" for a left branching, "r" for a right branching, from the tree root to the specific node. Table 21.3 summarizes the data subsets associated with these nodes, characterized by the chains of split conditions. The identification of these high-defect modules and their characterization can lead to focused remedial actions directed at such modules. These and other results were used by the development teams to guide their selective software inspection effort for cost-effective quality assurance and improvement.

As noted in Figure 21.3, tree-based modeling can handle categorical data or combined categorical and numerical data seamlessly — a unique advantage among all the risk identification techniques covered in this chapter. The treatment of these different types of data as predictor variables is similar except in defining binary partitions in the algorithm in Figure 21.3: cut off using "<" for an individual numerical predictor and binary grouping of its

Step 1. Both the dependent (response) variable and the explanatory (predictor or independent) variables are discretized by using cluster analysis or some other method if they are continuous.

Step 2. Select all statistically significant subsets defined by a pattern whose entropy (or uniformity) is within a threshold of the minimal entropy.

Step 3. Step 2 is repeated until no significant gain can be made in entropy reduction.

Figure 21.5 Algorithm for optimal set reduction

category values for an individual categorical predictors. The latter is more computationally intensive but still handled automatically in the tool S-PLUS. When the response variable is a numerical one, a regression tree similar to the examples earlier in this chapter and in Chapter 22 results. However, when the response variable is itself a categorical variable, the resulting tree is called a classification tree, as illustrated in Figure 21.7 (Section 21.5).

21.3.4 Pattern matching and optimal set reduction

In the above tree-based models, each subset of data can be uniquely described by a set of split conditions. Therefore, the data subset can be viewed as following a unique "pattern". However, many commonly defined patterns in practical applications do not have to be mutually exclusive, and they can be used in combination and in parallel to identify problematic areas. This kind of analysis can be carried out using a pattern matching technique called optimal set reduction (Briand et al., 1993).

The model construction for optimal set reduction can be summarized by the recursive algorithm in Figure 21.5. The *pattern* for a subset is defined by a condition on an explanatory (independent) variable, similar to the split conditions in tree-based models. The *entropy* is defined on the dependent variable values, capturing the uniformity of a subset. For example, for a subset of data, S, all mildly changed modules (subset S_1, characterized by the pattern: $1 \leq \texttt{CSI} \leq 10$, where CSI is the changed lines of code) are likely to have high defects (DF > 5, which defines the high-defect class). All modules with high data content (subset S_2, characterized by the pattern: operand-count > 50) are also likely to have high defects. Then, S_1 and S_2 can be extracted from S in parallel, because of the low entropy for these subsets (most of these modules are high-defect modules). Notice that these subsets may overlap, yielding a general graph instead of a tree structure, as illustrated by Figure 21.6.

Optimal set reduction was recently used to analyze various project effort and metrics data from NASA Software Engineering Laboratory, and to identify high-risk (high-effort) modules (Briand et al., 1993). It performed better (with 92.11% accuracy) than various other techniques, including classification trees, logistic regression without principal component, logistic regression with principal component (with 83.33%, 76.56%, and 80.00% accuracy, respectively). In addition, the combination of patterns for high-risk modules was also identified by the modeling result, which can be used to guide focused remedial actions.

21.4 COMPARISONS AND INTEGRATION

Similar to the cost–benefit analysis under different application environments for different quality assurance techniques in Chapter 17, we can compare the cost and benefit of individual

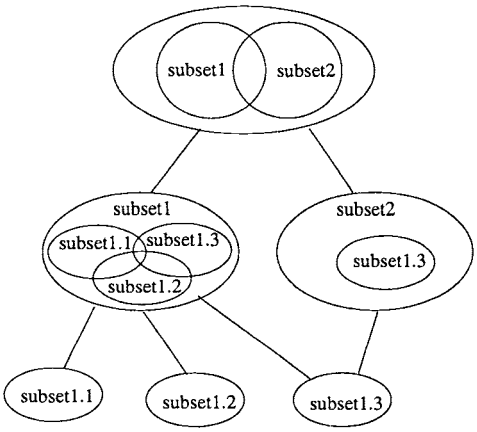

Figure 21.6 Example hierarchy for optimal set reduction

risk identification techniques. On the benefit side, the primary criteria include accuracy of the specific risk identification technique, the early availability of risk identification results and the related stability, and constructive information or guidance for quality improvement. These issues are individually examined below:

- *Accuracy* of analysis results can be measured by the difference (error) between predicted and actual results. The standard deviation of error can be used to measure accuracy for models with numerical response (for example, defect count), and proportion of misclassification for those with categorical response (for example, high-defect vs. low-defect). Since the data and applications are from diverse sources, only a qualitative comparison of result accuracy is possible here. In general, new techniques for risk identification discussed in Section 21.3 perform much better than traditional statistical techniques discussed in Section 21.2.

- *Early availability and stability:* There is a strong need for early modeling results, because problems found late in development are much harder and cost significantly more to fix. Ideally, models could be fitted to observations early and remain fairly stable so that timely and consistent remedial actions can be applied. All the techniques discussed in this chapter can be used early, but their stability differs considerably: Linear regression models are usually highly unstable due to high correlation in the metrics data, while models using principal component analysis are much more stable. On the other hand, techniques depending on data ranges (for example, tree-based modeling and optimal set reduction) are more likely to be stable than those depending on numerical values (for example, traditional statistical models).

- *Constructive information and guidance for quality improvement:* Tree-based models and optimal set reduction can characterize identified high-defect modules by their split conditions or patterns defined by certain metrics values or ranges. Such constructive information can be used to guide quality improvement activities. For example, if the identified high-defect subset of modules are characterized by numerous changes and high data contents, this information can be used in several ways: to minimize change for such modules, to reduce data contents by restructuring the modules, or to take extra precautions toward these modules.

Table 21.4 Comparison of risk identification techniques

Technique	Benefit/Performance			Cost/Usability		
	accuracy	stability	guidance	simplicity	interp.	tool sup.
correlation	poor	fair	fair	simplest	easiest	wide
regression	poor	poor	poor	simple	moderate	wide
discriminant	good	excellent	fair	moderate	moderate	moderate
neural net.	good	fair	poor	complex	hard	moderate
tree-based model	good	good	excellent	moderate	easy	moderate
opt. set reduction	good	fair	excellent	complex	moderate	limited

On the cost side, the primary cost item is the software professionals' time spent on performing the analysis, which can be affected by the complexity (or simplicity) of the technique itself and available tool support. Similarly, ease of result interpretation also affect the cost because of the possible time required not only to interpret the analysis results, but also to convince developers, tester, and managers to initiate follow-up activities for quality improvement. These issues are considered individually below:

- *Simplicity* of the analysis technique has many ramifications. A simple technique is generally easy to understand, easy to use, easy (and less costly) to perform on a given set of data, and is more likely to be supported by existing tools. Minimal amount of training is needed before a software quality professional can learn and master the technique. Among the risk identification techniques, correlation and regression analyses are simple statistical techniques, while others are more complex, with artificial neural networks (multiple parallel, hidden neurons) and optimal set reduction (overlapping subsets extracted in parallel) among the most complex.

- *Ease of result interpretation* plays an important role in model applications. A good understanding of the analysis results is a precondition to follow-up actions. For example, tree-based models present results in a form similar to decision trees commonly used in project management. Therefore, the results are easy to interpret and easy to use. On the other extreme, artificial neuron networks employ multiple hidden layer neurons and give the result as if from a "black-box", making result interpretation hard.

- *Availability of tool support* also has a significant influence on the practical applications of specific techniques. Traditional statistical analysis techniques covered in Section 21.2 are supported by many statistical packages. Some modern statistical packages also support principal component analysis, discriminant analysis, and tree-based modeling. However, special tools are needed to support artificial neural networks (several such tools are available) and optimal set reduction (a tool developed at the University of Maryland).

Table 21.4 summarizes our comparison of the risk identification techniques. Notice that principal component analysis is not listed as a separate entry, but rather included as part of discriminant analysis.

Like any other analysis techniques, the risk identification techniques are only a tool to provide us with evidence or symptoms of existing problems. The ultimate responsibility to

use the analysis results and to make changes lies with the development teams and their managers. Tree-based modeling technique seems to combine many good qualities appropriate for this kind of applications: It is conceptually simple, and is supported by a commercial tool S-PLUS. It provides accurate and stable results, and excellent constructive information, both in a consistent and uniform structure (tree) intuitive to the decision process. Therefore, tree-based modeling is an excellent candidate that can be used effectively to solicit changes and remedial actions from developers and managers. The analysis results and remedial action plans can often be cross-validated by other techniques, taking advantage of their individual strengths.

To facilitate practical applications of selected risk identification techniques, the analysis and follow-up activities need to be integrated into and carried out throughout the existing software development process. Such an integrated approach can be used to track quality changes and to identify and characterize problematic areas for focused remedial actions. This approach can be implemented in several stages: Initially, the analyses can be handled off-line by a dedicated quality analyst to minimize disruption to existing processes and to provide timely feedback. Thereafter, the analysis activities can be gradually automated, so that minimal effort is needed by the project teams to produce analysis results for remedial actions.

All the data and examples presented so far in this chapter are based on software artifacts, for example, defects and metrics data associated with specific software modules. However, similar risk identification techniques can also be applied to process or activity based data. For example, tree based modeling was used to link test results (successful vs. failed executions) to various timing and input state information for several IBM software products (Tian, 1995). The modeling results were used to identify clusters of test executions associated with abnormally high failures for focused remedial actions, which lead to significantly higher quality for these products as compared to earlier products. Similar analyses can also be performed on inspection data typically gathered in the earlier phases of software development. Each individual inspection can be treated as a data point, with all the circumstantial information associated with the inspection, such as components inspected, inspection method used, inspectors and time spent, as predictors, to build similar tree-based models to identify high-defect areas for focused quality improvement.

21.5 RISK IDENTIFICATION FOR CLASSIFIED DEFECT DATA

For classified defect data using ODC in Chapter 20, various analyses can be performed, such as the one-way and two-way analysis. However, the combinatorial explosion renders it impossible to perform multi-way analyses indiscriminately. We next examine the use of tree-based models to analyze such classified data and perform multi-way analysis, and illustrate it with examples for an IBM database product (Tian and Henshaw, 1994).

Defect impact analysis using TBM

For TBM analysis, we need to first select the response variable, and then examine its relationship to other variables used as predictor variables. In effect, we are performing 1-to-N analysis instead of isolated one-way analysis or the 1-to-1 two-way analysis. This 1-to-N analysis can be repeated for other variables if we select them as the response variable in individual analyses.

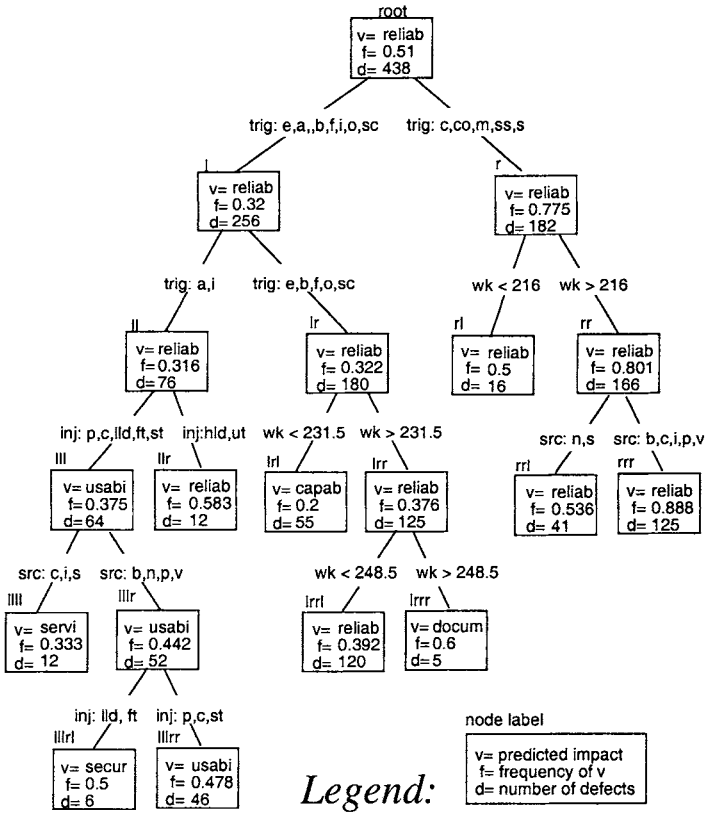

Figure 21.7 Predictions of defect impact for an IBM product

Defect impact is a classification of the defect itself, while all other defect attributes capture circumstantial information associated with the defect discovery and fixing. With the increased focus on customers in today's competitive environment, understanding of defect impact to customers takes on an increased importance. As a result, tree-based models to study the link between defect impact and other attributes would be of interests to project personnel, so that they can understand the linkage and devise appropriate corrective and preventive actions. The specific attributes used in the tree-based models were listed in Table 20.6.

Figure 21.7 shows the classification tree constructed for defect impact analysis. 438 defect entries and related ODC information were used after screening out data points with missing data. At each node (corresponding to a set of defects), the predicted defect impact v is the most frequently cited category by the testers for this set of defects. The analysis shows that the distribution of defects (defect impacts) is very non-homogeneous, and the relationship to other ODC attributes is highly nonlinear.

Detailed information associated with each tree node can be shown in stack-up barplot such as in Figure 21.8, where the vertical bar sequence of different shades represents the distribution of defect impact at this node. For example, the overall defect impact distribution among the impact areas is presented in the middle histogram stack in Figure 21.8. The left and right bars represent the defect impact distributions at nodes `l` and `r` of Figure 21.7 respectively.

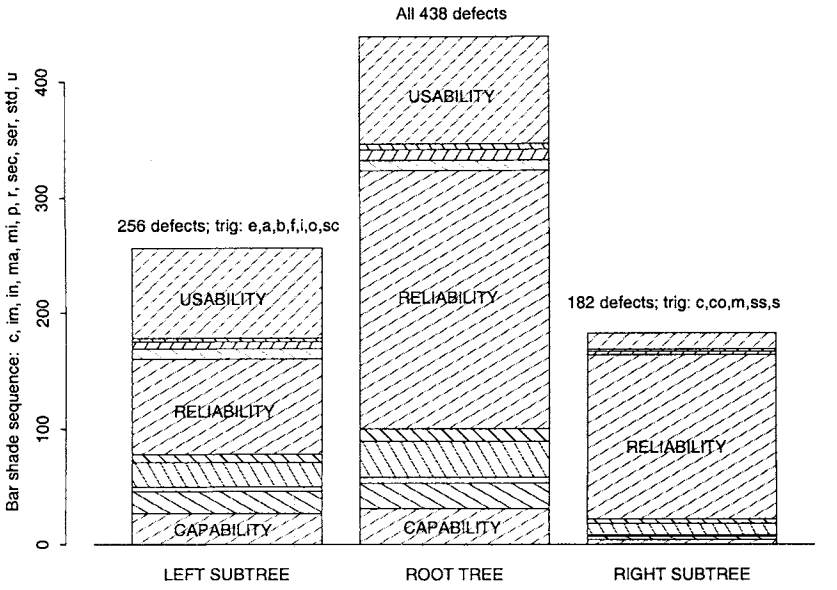

Figure 21.8 Defect impact distributions for an IBM product

Interpretation and usage of the analysis results

The primary partition for the defect impact is the defect trigger. After the partition, the reliability related defects become overwhelmingly dominant for the defects triggered by test cases from communication, coexistence, migration, stress, startup/shutdown scenario classes (`trig: c,co,m,s,ss`, see Table 20.6 in Chapter 20). For defects triggered by test cases from other scenario classes, although reliability remains a major problem (82 or 32% of the 256 defects), there is a disproportionate number of usability defects (78 or 30% of the 256 defects). The visual representation of this result in Figure 21.8 makes the impact of the primary partition obvious.

In the right sub-tree (rooted at node `r`), reliability impact is consistently predicted, although with different levels of f, which represents to some degree the confidence of the prediction. In the left sub-tree (rooted at node `l`), several other defect types are identified as the dominant defects in certain subsets. For example, usability defects are mostly triggered by testing scenarios of the ad-hoc and installation types. The defects were injected to base, new, refixed and vendor codes in the phases of coding, previous release, and system testing. This result can lead to a focused effort on the identified phases and types of code to remove the usability defects and enhanced test cases to thoroughly test for usability.

In part of the left sub-tree, data sets were partitioned into significantly smaller subsets to identify certain dominating defect impacts. For example, security defects dominate among the six defects associated with node `lllrl`, and documentation defects dominate among the five defects associated with node `lrrr`. In practical application, the information presented should only be used with both the set size d and frequency f in mind. Greater values for both f and d provide stronger evidence than smaller f or d.

The subsets of defects where certain impacts dominate can be easily identified in classification trees, together with their associated symptoms, by linking them to characteristics

used to navigate through the tree. Some of these symptoms may well be causes if confirmed or corroborated by developers and testers involved. In the context of the current analysis, there are essentially two ways to use the classification trees:

- *Passive tracking and occasional correction:* The classification trees can be used persistently over time to track problems, to identify certain high-risk or abnormal patterns of defect impact and associated symptoms, to guide additional causal analysis, and to derive appropriate remedial actions.
- *Active identification and control of product quality:* This use relies on externally identified targets of product quality and the identification of symptoms by classification trees. For example, if usability represents the primary concern for certain products, further analysis of usability-dominant subsets of defects needs to be done to actively identify and act upon corrective and preventive actions.

Many intricate links between data attributes can only be understood by the people with product-specific knowledge. Effective usage of modeling results requires close collaboration between the people from quality and process organizations and people intimately involved with the product development, testing, and data collection.

21.6 CONCLUDING REMARKS

Because of the highly uneven distribution of defects in software systems, there is a great need for effective risk identification techniques so that high-defect modules or software components can be identified and characterized for effective defect removal and quality improvement. The survey of different risk identification techniques presented in this chapter brings together information from diverse sources to offer a common starting point for software quality professionals and software engineering students. The comparison of techniques can help them choose appropriate techniques for their individual applications.

The tree-based modeling (TBM) technique was found to possess various desirable properties as an effective risk identification technique and is therefore highly recommended for various practical applications. Another example of the effective use of TBM for effective reliability risk identification and product reliability improvement is presented in Chapter 22. In addition, tree-based modeling can handle categorical data or combined categorical and numerical data seamlessly — a unique advantage among all the risk identification techniques covered in this chapter. Therefore, as a general recommendation, tree-based models should be considered in many situations for risk identification and quality improvement initiatives, either as the primary technique or to be integrated with other techniques to achieve the common goal of focused remedial actions on the identified problematic areas for effective quality improvement.

Besides the risk identification techniques based on empirical data we described in this chapter, there are various other techniques for risk identification, analysis, and management (Boehm, 1991; Charette, 1989). For example, software prototyping or rapid software prototyping can be used in software development or evolution to proactively identify and address various risks or potential problems (Luqi, 1989; Tanik and Yeh, 1989). Quantitative risk analyses based statistical decision theory (Pratt et al., 1965) can also be adapted to work with software prototypes to make management decisions based on related risk analysis (Cárdenas-García et al., 1992; Cárdenas-García and Zelkowitz, 1991). Risk identification also play a very important role in the spiral development process (Boehm, 1988) in assessing the project risk and initiating the next development spiral.