

Predicting field problems using metrics based models: a survey of current research

Paul Luo Li
Institute for Software Research International,
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh PA, 15232
412-268-3043
paul.li@cs.cmu.edu

ABSTRACT

Methods that can lower the cost of software field problems (e.g. faults, errors, failures, bugs, and defects) need field problem predictions. Models that predict field problems generally fall into two classes: time based models and metrics based models. In this paper, we examine metrics based models in detail. Metrics based models are better suited to predict field problems when an operational profile is not available, when the software and hardware configurations in use are unknown, and when the deployment and usage patterns are unknown. We present important concepts and the current state of research in inputs, output, and modeling methods.

Note to 654/754 students

You are only required to read sections 1-4. Also, this is a draft paper. So please excuse any mistakes (e.g. spelling mistakes and grammar mistakes), and let me know if you spot a mistake or feel uncomfortable about anything. Feedback is welcome and appreciated: Paul.Li@cs.cmu.edu.

1. INTRODUCTION

The US Department of Commerce estimates that field problems (e.g. faults, errors, failures, bugs, and defects) cost the U.S. economy an estimated \$59.6 billion dollars annually and that over half of the costs are borne by software consumers and the rest by software producers [78]. Field problem predictions may help lower the costs by guiding testing [45], improving maintenance resource allocation [69], adjusting deployment to meet the quality expectations of customers [74], planning improvement efforts [4], and enabling a software insurance system for software consumers [68].

Models that predict field problems generally belong to one of two classes: time based models and metrics based models [92]. In this survey, we briefly examine each class

of models. Then, we analyze metrics based models in detail.

Metrics based models can predict field problems using metrics available before release that capture various attributes of the software product, the development process, the deployment and usage pattern, and the software and hardware configurations in use.

We examine each component of metrics based models in detail: inputs, output, and modeling methods. This information can help practitioners decide how to implement a metrics based model for their projects and can help researchers decide where further research may be needed.

Section 2 discusses field problems. Section 3 reviews the different classes of models. Section 4 explains and discusses the current state of research for each component of metrics based models. Section 5 summarizes prior work. Section 6 is the conclusion.

2. FIELD PROBLEMS

We start by defining the observation of interest: field problems. The term *field problems* is intended to be generic and to encompass all the terms used in the literature to describe software related problems in the field.

Terms used in the literature to describe software related problems include faults, errors, failures, bugs, and defects. Different studies sometimes define these terms differently. Some studies use several terms interchangeably. To avoid confusion we use *field problems* to include all the terms. The only requirement is that the software related problem occurs in the field.

3. CLASSES OF MODELS

We use a classification scheme adapted from Schneidewind [92] and Tian [97] to divide models that predict software field problems into two classes:

1. Time based models: These models use the problem occurrence times or the number of problems in time intervals during testing to fit a software reliability model. The number of field problems is estimated by calculating the number of problems in future time intervals using the software reliability model.
2. Metrics based models: These models use historical information on metrics available before release (predictors) and historical information on software field problems to fit a predictive model. The fitted model and predictors' values for the current observation are used to predict field problems for the current observation.

The main differences between the two methods are the information used to make the predictions and the modeling assumptions. Time based models use the problem occurrence times or the number of problems in a time interval (time related problem information) during testing of the current observation as input. Metrics based models use a variety of metrics that capture different attributes the software system and the actual number of field problems from historical observations. Time based models assume that the problem occurrence pattern continues from testing into the field. Metrics based models do not assume a predefined relationship between predictors and field problems; instead, historical information on predictors and field problems is used to construct the models.

3.1 Time based models

Time based models assume that the software system has some probability of failure during every quantum of execution; therefore, a problem occurrence is a random process in time according to Musa et. al in [77]. This process is dictated by the number of residual problems and the discovery process (e.g. the amount of execution time). Prior work examining time based models assume that this random process can be modeled using a software reliability model. The idea is that every moment of execution has a chance of encountering one of the problems remaining in the code. The more problems there are in the code, the higher the probability that a problem will be encountered during execution. Assuming that a problem is removed once it is discovered, the probability of encountering a problem during the next execution decreases. Naturally, more problems will be found if more systems are executing the software system.

The major difference between different time based models is the model structures of the underlying software reliability models. The important form of the software

reliability models is the failure intensity function, which is defined by Lyu in [72] as the rate of problem occurrence at time t . Parameters of the models are usually estimated using time related problem occurrence information gathered during testing, methods like maximum likelihood, least squares, and method of moments, and a statistical computing program. The process is described in detail by Musa in [77]. The number of field problems is estimated by integrating the failure intensity function. The commonality between time based models is the use of time related problem occurrence information gathered during testing to fit a software reliability model and then predicting field problems using the fitted model. Farr discusses 17 different software reliability models in [72]. We present the exponential model as an example.

3.1.1 Exponential model

The exponential model is a widely used model, is one of the recommended models in the *AIAA Recommended Practice for Software Reliability* [1], and is discussed in detail by Musa et. al. in [77] and by Farr in [72].

The exponential model predicts the number of field problems using an exponential model. For example, assume that the defect discovery rate is 10 problems per unit time and 65 problems have been found up to the current time after 10 time intervals of testing. The failure intensity function, $\lambda(t)$, is then:

$$\lambda(t) = 107.01 * 10 * e^{-10 * t}$$

The function is plotted in Figure 2.

Let us assume that we release the software at the current time, $t=10$. Integrating the function from $t=10$ to infinity yields ~ 43 field problems

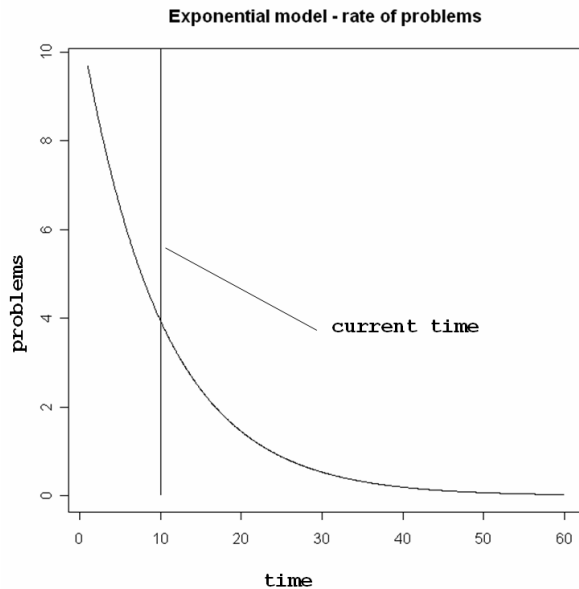


Figure 2. Failure intensity function for the exponential model

3.1.1.1 Limitations

Before talking about limitations of time based models, we define the operational profile, deployment and usage information, and hardware and software configurations information. Musa defines operational profile, deployment and usage, and hardware and software configurations in use in [72]. The operational profile is defined as the set of operations that the software can execute along with the probability with which they occur during operation. The software and hardware configurations in use are the hardware and software systems that interact with system during usage. Deployment and usage are the total number of deployed systems and the amount of execution of the systems.

In order for the defect occurrence pattern to continue into future time intervals, the software has to be operated in a similar manner as that in which reliability predictions are made. The similarity of testing and deployment environments assumption is one of the key assumption for time based models cited by Farr in [72]. To extend the software reliability model from testing to the field, an accurate operational profile, similar hardware and software configurations, and information on deployment and usage are required.

The information is available in certain situations such as Navy projects at McDonnell Douglas studied by Jelinski and Moranda in [29] and NASA projects studied by Schneidewind in [93]. However, for other types of systems, such as the commercial systems, the operational profile, information on deployment and usage, and information on hardware and software configurations in

use may be unattainable or may contain too many scenarios to be tested compressively.

When the similarity of testing and deployment environments assumption is broken, it is usually not possible to extend the software reliability model fitted using development problems into the field. For example, Kenny and Li et. al. examined three commercial systems developed by IBM in [34] and [69]. All the systems examined exhibited initially increases in the rate of field defect occurrences. A software reliability model extended from development cannot describe the observed patterns of field defect occurrences. Li et. al. show in [70] that a strictly decreasing software reliability model, e.g. the exponential model, cannot model an increasing rate of defect occurrences. Kenny shows in [35] that it is not possible to model the increasing defect occurrence pattern using a Weibull model assuming that the rate of defect occurrences is decreasing at the time of release (i.e. the software has been properly tested).

3.2 Metrics based models

Metrics based models can use metrics that capture attributes of the software product, the development process, deployment and usage, and software and hardware configurations in use available before release (predictors) to predict field problems; therefore effects of various attributes on field problems can be explicitly accounted for in the models. The idea is that certain characteristics make the presences of field problems more or less likely. Capturing the relationship between these characteristics and field problems using past observations allows field problems to be predicted for unforeseen observations.

Metrics are defined by Fenton and Pfleeger in [16] as outputs of measurements, where measurement is defined as the process by which values are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.

Unlike time based models, metrics based models use historical information on predictors and the actual number of field problems to construct the predictive model. Different metrics based models use different modeling methods to model the relationship between predictors and field problems. Since there is no assumption about the similarity between testing and field environments, metrics based models are more robust against differences between how the software is tested and how it is used in the field.

4. METRICS BASED MODELS

We examine each component of metrics based models in this section. Section 4.1 examines the inputs. Section 4.2

examines the output. Section 4.3 examines the modeling methods.

4.1 Inputs

The inputs to metrics based models are metrics' values. We categorize metrics used in literature using an augmented version of the categorization schemes used by Fenton and Neil in [18], Khoshgoftaar and Allen in [37], and the IEEE standard for software quality metrics methodology [27]:

- **Product metrics:** metrics that measure the attributes of any intermediate or final product of the software development process [27]. The product metrics in the literature are computed using a snapshot of the code. There are tools compute product metrics automatically, such as the EMERALD [32], COSMOS [13], and Logiscope [85]. Product metrics have been shown to be important predictors by studies such as Khoshgoftaar et. al. [45], Takahashi et. al. [96], Jones et. al. [32], and Shelby and Porter [95].
- **Development metrics:** metrics that measure attributes of the development process. The development metrics in the literature are usually computed using information in version control systems and change management systems. Development metrics have been shown to be important predictors by studies such as Khoshgoftaar et. al. [50], Harter et. al. [25], and Shelby and Porter [95].
- **Deployment and usage (DU) metrics:** metrics that measure attributes of the deployment of the software system and usage in the field. Few studies have examined deployment and usage metrics, and no data source is consistently used. DU metrics have been shown to be important predictors by studies such as Jones et. al. [32], Khoshgoftaar et. al. [51], Khoshgoftaar et. al. [64], Mockus et. al. [74].
- **Software and hardware configurations (SH) metrics:** metrics that measure attributes of the software and hardware systems that interact with the software system in the field. Few studies have examined SH metrics and no data source is consistently used. SH metrics have been shown to be important predictors by Mockus et. al. [74].

Product, development, deployment and usage, and software and hardware configuration metrics available before release are *predictors*, which are used to predict field problems.

- **Field problems:** metrics that measure field problems. Field problem metrics in literature are usually computed using information in change managements systems and defect tracking systems. Each study has at least one field problem metric. Field problem metrics include the number of faults, bugs, errors, and defects and are discussed in section 2.

The definitions of specific metrics (i.e. rules for counting) may differ slightly between studies, which makes comparison and evaluation of metrics difficult. This is a well known problem and is discussed in detail by Fenton and Pfleeger in [16].

For example, consider the following example examining the differences between a widely-used definition of failures (e.g Lyu in [72] and Zhu et. al. in [108]) by Laprie [67], and a definition of defects by Li et. al. [69].

Laprie describes failures in [67]. A failure is a deviation between the delivered service and the specified service, where the service specifications are an agreed description of the expected service.

Li et. al defines a defect as a user-reported problem that requires developer intervention to correct [69]. Examples of defects include APARs (Authorized Program Analysis Report), which are customer reported problems that require code change recorded by IBM development organizations and on-line bug reports, which are user-reported problems that require a developer's action to resolve recorded by open source software projects [69].

Subtle differences exist between a failure and a defect as defined above. A defect may not be counted as a failure if the software system lacks specifications or if the specifications are incomplete, as discussed by Chillarege in [72]. A failure may not be counted as a defect if the user does not report the failure.

Similar problems can occur when two studies report collecting the same metric. Different studies can report collecting the same metric but are applying different counting rules. This is discussed by Ohlsson and Runeson in [85]

In our survey, we attempt to use the most widely accepted definition of a metric when necessary and to avoid differing definitions wherever possible. The idea is to examine the intent of the metric and not the instantiation of the metric in any particular setting.

In this section we examine each category of predictors, the metrics collection process, and methods of showing a metric is important.

4.1.1 Product metrics

The obvious place to look for attributes that may be related to software field problems is in the software product itself. Product metrics are the most widely used metrics in the studies we survey.

Munson and Khoshgoftaar identify dimensions (i.e. source of variation) within product metrics in the literature in [76]. Many of the product metrics used in the literature measure similar things and are highly correlated with each other (e.g. lines of code and source lines of code) as discussed by Fenton and Neil in [18]. Using principal component analysis, Munson and Khoshgoftaar identify metrics that capture the same intent (i.e. the same dimension) and attempt to describe the dimensions. Principal component analysis is an analysis method that creates linear combinations of a set of predictors to encapsulate the maximum amount of variation in the dataset and is orthogonal (i.e. uncorrelated) to the other principal components [76]. By examining the loading (i.e. how much a predictor contributes to a principal component) it is possible to see which predictors capture the same source of variation. The dimensions of product metrics identified by Munson and Khoshgoftaar are:

- Control: metrics related to the control flow complexity. Examples are Cyclomatic complexity and the number of nodes in the control graph (refer to [73] for a detailed explanation of the metrics).
- Volume: metrics related to the number of distinct operations and statements. Examples are number of unique operands and source lines of code (refer to [24] for a detailed explanation of the metrics).
- Action: metrics related to the number of operations or operators in the program. Examples are unique operators and source code statements (refer to [24] for a detailed explanation of the metrics).
- Effort: metrics related to the mental effort required to generate an implementation from a specification. Examples are Halstead's program effort metrics (refer to [24] for a detailed explanation of the metrics).
- Modularity: metrics related to the degree of modularization of a program. Examples are the number of function calls and the number of statements at a nest level of 10 or greater (refer to [76] for a detailed explanation of the metrics).

4.1.2 Development metrics

Since the software product is the result of the software development process, the next logical place to look is in the development process. The intuition behind

development metrics is that attributes of the development process (i.e. how the product is implemented) is related to field problems.

No study has yet identified the dimensions in development metrics. We present a rough grouping of the development metrics in the literature based on the description of the metrics:

- Problems discovered prior to release: metrics that mention measuring attributes of problems found prior to release in the description. Examples are number of field problems in the prior release used by Ostrand et. al. [87], number of development problems used by Fenton and Ohlsson [17], and number of problems found by designers used by Khoshgoftaar et. al. [62].
- Changes to the product: metrics that mention measuring attributes of changes made to the software product in the description. Examples are reuse status used by Pighin and Marzona [88], changed source instructions used by Troster and Tian [99], number of deltas (changes to the code) used by Ostrand et. al. [87], and increase in lines of code used by Khoshgoftaar et. al. [63].
- People in the process: metrics that mention measuring attributes of people involved in the development process in the description. Examples are the number of different designers making changes and the number of updates by designers who had 10 or less total updates in entire company career both used in Khoshgoftaar et. al. [64].
- Process efficiency: metrics that mention measuring attributes of the maturity of the development process or the effort expended on the development process in the description. Examples are CMM level used by Harter et. al. [25] and total development effort per 1000 executable statements used by Selby and Porter [95].

4.1.3 Deployment and usage metrics

The intuition behind deployment and usage metrics is the same idea behind operational profiles. The amount of execution and the kinds of execution during operation are related to field problems. Only two distinct research efforts consider deployment and usage metrics in the papers we survey: one by Khoshgoftaar et. al. and one by Mockus et. al..

Khoshgoftaar et. al. consider the following deployment and usage metrics for modules in e.g. [102], [49], [50], [40], [64], and [103]:

- Proportion of systems with a module installed
- Execution time of an average transaction on a system serving customers
- Execution time of an average transaction on a systems serving businesses
- Execution time of an average transaction on a tandem system

The execution times are calculated by running the systems using an operational profile. The proportion of systems with module installed is derived using deployment records [102].

Mockus et. al. consider the following deployment and usage metrics for installations of a telecommunications software system in [74]:

- Number of ports on the customer installation
- Total deployment time of all installations in the field at the time of installation

The number of ports is computed using information in a customer hardware database. The total deployment time of all machines in the field is computed from customer deployment records [74].

The intent of each metric above is to capture information about the amount or the kinds of execution in the field. However, the data sources used to capture the metrics may not be available for all systems. In addition, no data source has emerged as a reliable source of deployment and usage metrics.

4.1.4 Software and hardware configurations metrics

The intuition behind software and hardware configurations metrics is that some field problems can only be exposed using certain configurations; therefore, the software and hardware configurations in use are related to field problems. Only one research efforts consider software and hardware metrics in the papers we survey. The paper is by Mockus et. al..

Mockus et. al. consider the following software and hardware configuration metrics for installations of a telecommunications software system in [74]:

- Systems size of the installation (the hardware that is associated with large or small/medium sized installation)
- Operating system of the installation (proprietary, Linux, or Windows)

The system size and operating system are computed using information derived using deployment records [102].

The intent of each metric above is to capture information about the software and hardware configurations in use in the field. However, the above information may not be available for all systems. No data source has emerged as a reliable source of software and hardware configuration metrics.

4.1.5 Metrics collection

The literature shows no agreement on which specific metrics are “good” metrics as demonstrated by the continuing debate by Kitchenham et. al. in [65] and by Weyuker in [107]. Despite disagreement on which specific metrics to collect, there is general agreement on the need for more metrics that capture different attributes as stated in the IEEE standard for software quality metrics methodology [27]. Prior work shows that, in general, collecting and using more metrics will result in more accurate field problem predictions and that metrics in each category above is important.

The general approach is to collect all reasonable metrics that are consistent for all observations within the study. Prior work generally collects metrics that measure attributes that can be reasoned as being related to field problems and are measured in the same manner for all observations. This avoids spurious correlations and ensures that the relationships discovered will be reasonable for the particular setting as discussed in [74] and [16]. Furthermore, IEEE [27] recommends that each organization perform a cost-benefit analysis to assess how many metrics to collect and which metrics are appropriate.

4.1.6 Methods of showing a metric is important

Prior work shows that product, development, deployment and usage, and software and hardware configurations metrics are all important. Due to differences in the exact definitions of specific metrics and differences in the metrics collected between studies, we examine categories of metrics.

To show that a category of metrics is important, it is sufficient to show that a predictor in the category is important. There are generally four ways of showing that a predictor is important:

1. Show high correlation between the predictor and field defects. This method is recommended by IEEE [27] and is used by Ohlsson and Alberg [83] and Ostrand and Wyuker [86].
2. Show that the predictor is selected using a model selection method. This method is used by Harter et. al. [24] and Mockus et. al. [74].

3. Show that the accuracy of predictions improves with the predictor included in the prediction model. This method is used by Khoshgoftaar et. al. [46] and Jones et. al. [32].

An example of method 1 is in Ohlsson and Alberg [4]. The authors compute the correlations between product predictors and the number of field problems. The authors select predictors that have a correlation higher than .4 as important. In the paper, 11 out of 27 predicted are selected using this method. The three highest correlated metrics and the correlations are shown in table 1.

Table 1. Correlations from Ohlsson and Alberg [4].

Predictor	Correlation (r)
SigFF: Number of new or modified calls	.64
McC1: Cyclomatic complexity number	.54
McC2: Modified Cyclomatic complexity number that does not punish for higher modularization	.48

An example of method 2 is in Harter et. al. [24]. The authors use a linear regression model to predict the number of errors. The authors use the p-value of the estimated parameter value to select important predictors. The summary of the linear regression model is in Table 2. The development metric is process maturity measured by the CMM level. The p-value associated with its parameter estimate is 0; therefore the product metric is a significant predictor at the 99% confidence level.

Table 2. Results from Harter et. al. [24]

Variable	Parameter	Estimated value using least squares
Intercept		
	β_0	5.597
	s.e	.464
	T	12.059
	P	0.000

ln(Process Maturity) CMM level	B_1	1.589
	Se	.386
	T	4.116
	P	0.000
ln(Product size) lines of code	β_2	.234
	s.e	.108
	T	2.160
	P	0.020
ln(Product-Design-Complexity) subjective evaluation	β_3	-2.11
	s.e	.712
	T	-2.963
	P	0.003

An example of method 3 is in Jones et. al. [32]. The authors construct two logistic models that classify modules as risky (will experience a field problem) and not risky (will not experience a field problem). One model uses only product metrics and the other uses product metrics and a deployment and usage metric. The authors show that the model with the deployment and usage metric has lower type II errors for the testing set. The authors argue that since identifying risky modules (i.e. not making a type II error) is more important, the model with the deployment and usage metric is better; therefore, deployment and usage metrics are important.

FILINCQU: number of distinct include files

LGPATH: log base 2 of the number of independent paths in the flow graph

VARSPMAX: maximum span of variables (statements between declaration and use of a variable)

USAGE: proportion of systems with module installed

$$\text{Logit(Faults)} = -5.13 + .0284 \text{ FILINCQU} + .0209 \text{ LGPATH} + .00043 \text{ VARSPMAX}$$

Each predictor is significant at the 15% level

Type I error = 27.32%
 Type II error = 34.24%

$$\text{Logit}(\text{faults}) = -5.13 + .0284 \text{ FILINCUQ} + .0209 \text{ LGPATH} + 1.2718 \text{ USAGE} + .00043 \text{ VARSPNMX}$$

Each predictor is significant at the 15% level
 Type I error = 29.06%
 Type II error = 30.77%

We present findings from studies that use method 3 in Table 3. A level 1 output is a prediction of whether an observation will be risky (e.g. have a field problem) or not risky (e.g. will not have a problem). The measures of accuracy for a level 1 output include the type I error, which measures the proportion of observations predicted as risky when it is actually not risky, the type II error, which measures the proportion of observations predicted as not risky when it is actually risky, and the overall error, which measures the overall proportion of misclassified observations. A level 2 output is a prediction of the number of field problems. The measures of accuracy for a level 2 output include the average relative error (ARE), the average absolute error (AAE). Types of output and measures of accuracy are discussed in detail in section 4.2.

Table 3. Changes in accuracy with additional categories of metrics

Research work	Type of output	Category of metric examined	Other categories of metrics in model	Accuracy of model without the category of metric	Accuracy of model with the category of metric
Khoshgoftaar et. al. [48]	Level 1	Development	Product	26.0% type I error 18.75% type II error 25.2% overall error	24.8% type I error 15.0% type II error 23.6% overall error
Khoshgoftaar et. al. [45]	Level 1	Development	Product	32.4% type I error 21.3% type II error 31.1% overall error	23.8% type I error 13.8% type II error 22.6% overall error
Khoshgoftaar et. al.[50]	Level 1	Development	Product	27.0% type I error 27.4% type II error	26.2% type I error 28.9% type II error
Ostrand et. al. [87]	Level 1	Development	Product	37% type II error	16% type II error
Jones et. al. [32]	Level 1	Deployment and usage	Product	27.32% type I error 34.24% type II error	29.06% type I error 30.77% type II error
Khoshgoftaar et. al. [51]	Level 1	Deployment and usage	Product Development	23.55% type I error 32.80% type II error	30.30% type I error 23.81% type II error
Khoshgoftaar et. al. [64]	Level 1 over multiple releases (release 2)	Development	Product Deployment and usage	26.6% type I error 24.9% type II error	29.3% type I error 21.2% type II error
Khoshgoftaar et. al. [64]	Level 1 over multiple releases (release 3)	Development	Product Deployment and usage	28.8% type I error 21.3% type II error	29.9% type I error 19.1% type II error

Khoshgoftaar et. al. [64]	Level 1 over multiple releases (release 4)	Development	Product Deployment and usage	32.7% type I error 27.2% type II error	32.7% type I error 19.6% type II error
Khoshgoftaar et. al. [62]/[63]	Level 1 over multiple releases (release 2)	Development	Product Deployment and usage	24.76% type I error 25.93% type II error	25.32% type I error 23.81% type II error
Khoshgoftaar et. al. [62]/[63]	Level 1 over multiple releases (release 3)	Development	Product Deployment and usage	28.25% type I error 29.79% type II error	27.36% type I error 19.15% type II error
Khoshgoftaar et. al. [62]/[63]	Level 1 over multiple releases (release 4)	Development	Product Deployment and usage	35.59% type I error 21.74% type II error	25.71% type I error 27.17% type II error

4.2 Output

This section examines the output of metrics models (i.e. what is predicted about field defects). Different results may allow different action to be taken to reduce the cost of field problems. Research work generally produced three levels of output (results) shown in Table 4. Rest of this section discusses each level of output and the experimental set up to evaluate an output.

Table 4. Output of papers

Level	Output	Research question addressed	Research work
Level 0	A relationship	What predicts field problems?	Basili and Perricone [3] Bassin and Santhanam [4] Fenton and Ohlsson [17] Harter et. al. [24] Ohlsson and Wohlin [84] Ostrand and Weyuker [86] Pighin and Marzona [88] Troster and Tian [99]
Level 1	A categorical	Is it risky or not? (Is	Briand et. al.[5]

output	the number of field defects above a threshold?)	Ebert [13]/[14]/[15] Jones et. al. [32] Karathanithi [36] Khoshgoftaar and Allen [37] Khoshgoftaar and Allen [38] Khoshgoftaar et. al. [39] Khoshgoftaar et. al. [40] Khoshgoftaar et. al. [41] Khoshgoftaar et. al. [42] Khoshgoftaar et. al. [43]/[44] Khoshgoftaar et. al. [45] Khoshgoftaar et. al. [48] Khoshgoftaar et. al. [49] Khoshgoftaar et. al. [50] Khoshgoftaar et. al. [51] Khoshgoftaar et. al. [53] Khoshgoftaar et. al. [54]/[55] Khoshgoftaar and Seliya
--------	---	--

			<p>[59] Khoshgoftaar and Seliya [61] Khoshgoftaar et. al. [62]/[63] Khoshgoftaar et. al. [64] Kokol et. al. [66] Mockus et. al. [74] Munson and Khoshgoftaar [75] Ohlsson and Runeson [85] Ostrand et. al. [87] Pighin and Zamolo [89] Pighin et. al. [90] Schenker and Khoshgoftaar [91] Selby and Porter [94] Selby and Porter [95] Takahashi et. al. [96]</p>
Level 2	A numerical output	What is the number of field problems?	<p>Graves et. al. [22] Khoshgoftaar et. al. [46] Khoshgoftaar et. al. [52] Khoshgoftaar et. al. [56] Khoshgoftaar et. al. [57] Khoshgoftaar et. al. [58] Khoshgoftaar and Seliya [60] Xu et. al. [102] Yuan et. al. [103]</p>

4.2.1.1 Level 0 result

Prior work at level 0 establishes relationships (sometimes with models) between predictors' values and the value of the field problem metric. Results are relationships between predictors and field problems.

A level 0 result may allow for improvement planning, better allocation of maintenance resources, or improvement of testing efforts. Harter et. al. [24] and Bassin and Santhanam [4] evaluate the effectiveness of the development process for improvement planning. Harter et. al. evaluate the development process by examining the CMM level of the organization. Bassin and Santhanam evaluate the development process by examining the distribution of ODC triggers of problems found during development.

Determining that a predictor is important may allow for better allocation of maintenance resources and improved testing. For example, Mockus et. al. establish the

relationship between the operating systems platform (i.e. a proprietary OS, Linux, and Windows) and field problems in [74]. In addition to allowing better testing of the fault prone platforms, this may also allow the right maintenance personnel (i.e. personnel with knowledge of the right operating system) to be staffed to address the field problems.

Methods of evaluating which predictors are important are discussed in section 4.1.1.5. A study that produce level 1 or level 2 result automatically include a level 0 result; since in order to have a level 1 or level 2 result, a model must be first fitted. Some studies that only report a level 0 simply observe a correlation between the predictors' values and the values of the field problems metric and appeal to reason (e.g. Ostrand and Weyuker [86] and Fenton and Ohlsson [17]).

4.2.1.1 Level 1 result

Prior work at level 1 establishes relationships between predictors' values and the class of the field problem metric using models, then uses the models to classify observations into one of two classes: risky or not risky. Results are classifications.

The primary purpose of a level 1 result is to focus testing efforts on risky modules. First we discuss how to evaluate a level 1 output.

The most commonly used measures of accuracy of a level 1 output are type I error, type II error, and overall error. We use the definitions by Ohlsson and Runeson in [85]. A type I error occurs when an observation is classified as risky when the observation is actually not risky (i.e. a false positive). A type II error occurs when an observation is classified as not risky when the observation is actually risky (i.e. a false negative). Some papers may reverse the definitions of type I error and type II error. Overall error is the overall rate of misclassification.

Determining which observation is risky may allow testing effort to be focused in the appropriate places. This focus discussed in detail by Selby and Porter [94] and Khoshgoftaar et. al. [54]. In general, type II errors are more important, because the main objective of classifying observations is to reduce the cost of field problems by removing problems before the software system is deployed as cited by Jones et. al. in [32]. However, since resources are limited, high type I errors and overall errors are also not desirable. Only a selected number of observations can be chosen for additional testing. The costs of misclassification need to be considered in each setting to select an optimal balance as discussed by Khoshgoftaar et. al. in [49]. A level 1 output allows the decision to be based on quantitative results.

For example, consider the data set in table 5 in which the top 40% of observations ranked according to the number of field defects are classified as risky. The same kind of approach is taken by Munson and Khoshgoftaar in [75]. Risky is encoded as 1. Not risky is encoded as 0.

Table 5. Example classification

Obs	Predicted field problems	Predicted class	Actual field problems	Actual class
1	.7	0	1	0
2	1.32	0	4	1
3	1.52	0	0	0
4	2.07	0	0	0
5	2.12	0	0	0
6	2.34	1	4	1
7	2.67	1	2	0
8	2.98	1	6	1
9	3.12	1	1	0
10	3.67	1	3	1

Two observations are predicted as risky when they are not risky (observations 7 and 9). The total number of not risky observations is 6. This results in a 33.33% type I error. One observation is classified as not risky when it is risky (observation 2). The total number of risky modules is 4. This results in a 25% type II error. The overall misclassification rate is 3 observations out of 10, which results in a 30% overall error. The classification errors are summarized in Table 6.

Table 6. Summary of classifications

	Predicted not risky	Predicted risky	Total
Actual not risky	4 66.67%	2 33.33%	6
Actual risky	1 25%	3 75%	4
Total	5	5	10

4.2.1.2 Level 2 output evaluation

Prior work at level 2 establishes relationships between predictors' values and the value of the field problem metric using a model, and then uses the model to quantify the risk. Results are predicted values of the field problem metric.

Determining the number of field problems may allow the appropriate amount of maintenance resources to be

allocated; in addition, as shown by the example in section 4.2.1.2, it is also possible to use a level 2 output to determine where to focus testing by selecting a percentage of the observations. Not having sufficient resources may delay field problem resolution, which results in reduced customer satisfaction as shown by Chulani et. al. [10]. Allocating too many resources hinders other efforts (e.g. development). Therefore, allocating the correct amount of resources is important. Having a level 2 result and knowing the errors associated with the predictions are steps towards quantitatively based decision making [74].

The most commonly used measures of accuracy of level 2 output are the average relative error (ARE), the average absolute error (AAE), the standard deviation of the relative errors, and the standard deviation of the absolute errors. The AAE measures the average error in predictions (i.e. how much a typical prediction will be off by). The AAE can be misleading when the predicted number of field problems differs significantly between observations; therefore, ARE is often reported as well. The ARE measures the average percentage of error in the predictions (i.e. relative to the actual number of field problems, how much a typical prediction will be off by).

The average absolute error is defined by Khoshgoftaar et. al. in [56] as the sum over all observations, the absolute value of the difference between the predicted value and the actual value.

\tilde{y}_i = predicted number of field problems

y_i = actual number of field problems

$$AAE = 1/n \sum_{i=1}^n |(\tilde{y}_i - y_i)|$$

Absolute relative error is defined by Khoshgoftaar et. al. in [56] as the sum over all observations, the absolute value of the difference between the predicted value and the actual value divided by the actual value plus one. The denominator of the ARE has one added to avoid dividing by zero.

$$ARE = 1/n \sum_{i=1}^n |(\tilde{y}_i - y_i) / (y_i + 1)|$$

The standard deviation of the relative error and standard deviation of the absolute error are the standard deviation of the relative error of the observations and the standard deviation of the absolute error of the observations respectively.

For example, consider the set of predictions in Table 4. On average, each prediction is off by 86.10%. On average, each prediction is off by 1.683 ~ 2 field problems.

$$AAE = 1/10 (0.30 + 2.68 + 1.52 + 2.07 + 2.12 + 1.66 + 0.67 + 3.02 + 2.12 + 0.67) = 1.683$$

The standard error AE is: 0.901

$$ARE = 1/10 (.15 + .536 + 1.52 + 2.07 + 2.12 + .332 + .223 + .431 + 1.06 + .1675) = 0.86095.$$

The standard error of RE is: .7806

Table 4. Example predictions

Obs	Predicted number of field problems	Actual number of field problems
1	.7	1
2	1.32	4
3	1.52	0
4	2.07	0
5	2.12	0
6	2.34	4
7	2.67	2
8	2.98	6
9	3.12	1
10	3.67	3

4.2.2 Experimental setup for evaluation

Evaluating predictions (i.e. level 1 and level 2 outputs) involves fitting a prediction model then evaluating predictions for unseen observations. There are two common ways of setting up this evaluation in the literature. One is the holdout method (i.e. having separated training and testing data sets) described by Ebert in [14]. The other is cross-validation (i.e. repeatedly withholding part of the data, fitting the model, predicting for the withheld observations, and evaluating the predictions) described in Selby and Porter [94].

Each method has its drawbacks. It may not be possible to use the holdout method if there is only a limited amount of data. Also, there is the possibility that the training set is biased (i.e. an “unfortunate” sample in which anomalous observations are selected to be the training set), which will result in an inaccurate model. With the cross-validation technique, the estimated error rate will be higher or the variance of the estimated error will be larger. In addition, the cross-validation technique is more computationally expensive. The tradeoffs between the holdout method and the cross-validation are discussed in detail in Venables and Ripley [100].

4.3 Modeling methods

Modeling methods are ways to produce models using historical information on predictors’ values and field problem metric values such that the resulting model can produce a prediction given the predictors’ values for a new observation (we will not examine level 0 outputs). We will examine modeling methods by the kind of output they produce:

- Level 1 output (section 4.3.2):
 - Linear modeling (logistic regression)

- Trees
- Discriminant analysis
- Rules
- Neural networks
- Clustering
- Sets
- Linear programming
- Heuristics or any level 2 method with heuristics
- Level 2 output (section 4.3.3):
 - Linear modeling (linear regression and negative binomial regression)
 - Non-linear regression
 - Trees
 - Neural networks

The exceptions are principal component analysis, bagging, boosting, and logitboost, and fuzzy logic. Principal component analysis takes the predictors as input and outputs a set of new predictors. This technique is described in section 4.3.4. Bagging, boosting, and logitboost take results from multiple runs of a modeling technique to decide upon an output. These techniques are described in section 4.3.5. Fuzzy logic accounts for uncertainty in values by assigning probability to values. This technique is discussed in detail in section 4.3.6. We discuss combination of techniques in section 4.3.7. We provide a partial ordering of modeling methods in section 4.3.8. We examine other methods of evaluating modeling methods in section 4.3.9. First we present an example.

4.3.1 Example: the trees technique

We illustrate the model construction process and the prediction process using the trees technique. The trees technique is the most popular modeling technique in the literature

According to Selby and Porter in [94], the trees technique involves creating partitions in the observations based on predictors’ values that minimizes the error in classifications within the partitions. The process is repeated until the error within each partition is below some limit or until the number of observations within each partition is below some limit. The most important predictors are automatically selected, and the trees technique is distribution independent (i.e. does not require errors to be normally distributed).

We illustrate the construction and use of a trees model to classify modules as risky and not risky.

While minimum error or minimum observation is not reached for all partitions, first generate candidate partitions using predictor values, then partition the data using the predictor value that minimizes error.

Consider the following simple example:

Predictor A has three values: 1, 2, 3

Predictor B has two values: 1, 2

The field problem metric has two classes (values): 1 (at least 1 field problem), 0 (no field problems)

The measure of error is: $\sum_{\text{partitions}} \sum_{\text{all observations in partition}} |y_i - \tilde{y}|$

\tilde{y} = mean of classifications in the partition

The minimum error in partition: 0

The minimum number of observation: 2

We use the training set in Table 7.

Table 7. Training set

Obs	Value of Predictor A	Value of Predictor B	Class of the field problems metric
1	1	1	0
2	1	2	0
3	1	1	0
4	1	2	0
5	2	1	1
6	2	1	1
7	3	1	0
8	3	2	0
9	3	1	0
10	3	2	1

Iteration 1

Minimum error or minimum observation not reached for all partitions.

Generate candidate partitions using predictors' values:

- A <=1
 - error in partition 1 (A<=1)
 - (0 + 0 + 0 + 0) = 0
 - error in partition 2 (A>1)
 - (1/2 + 1/2 + 1/2 + 1/2 + 1/2 + 1/2) = 3
 - total error = 3
- A <=2
 - error in partition 1 (A<=2)
 - (1/3 + 1/3 + 1/3 + 1/3 + 2/3 + 2/3) = 2.667
 - error in partition 2 (A>2)
 - (1/4 + 1/4 + 1/4 + 3/4) = 1.5
 - total error = 4.167
- B <= 1
 - error in partition 1 (B<=1)
 - (1/3 + 1/3 + 1/3 + 1/3 + 2/3 + 2/3) = 2.667

- error in partition 2 (B>1)
 - (1/4 + 1/4 + 1/4 + 3/4) = 1.5
- total error = 4.167

Based on total error, partition using A<=1. The resulting tree is in Figure 4. Since the error in partition A<=1 is 0, only observations in the partition A>1 (observations 5-10) are examined in the next iteration.

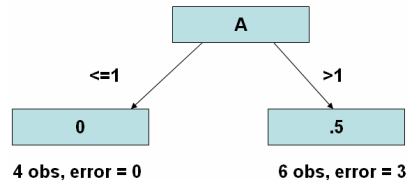


Figure 4. Tree after one iteration

Iteration 2

Minimum error or minimum observation not reached for all partitions.

Generate candidate partitions using predictors' values:

- A <=1: not possible
- A <=2
 - error in partition 1 (A<=2)
 - (0 + 0) = 0
 - error in partition 2 (A>2)
 - (1/4 + 1/4 + 1/4 + 3/4) = 1.5
 - total error = 1.5
- B <= 1
 - error in partition 1 (B<=1)
 - (1/2 + 1/2 + 1/2 + 1/2) = 2
 - error in partition 2 (B>1)
 - (1/2 + 1/2) = 1
 - total error = 3

Based on total error, partition using A<=2. The resulting tree is in Figure 5. Since the error in partition A<=1 and partition A<=2 is 0, only observations in the partition A>2 (observations 7-10) are examined in the next iteration.

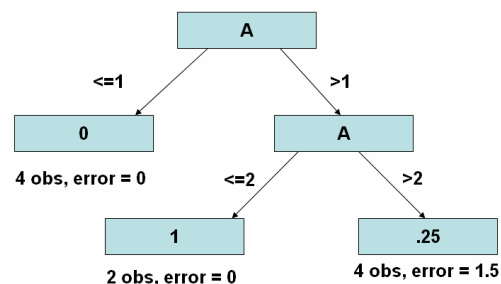


Figure 5. Tree after two iterations

Iteration 3

Minimum error or minimum observation not reached for all partitions.

Generate candidate partitions using predictors' values:

- A <=1: not possible

- $A \leq 2$: not possible
- $B \leq 1$
 - error in partition 1 ($B \leq 1$)
 - $(0+0) = 0$
 - error in partition 2 ($B > 1$)
 - $(1/2 + 1/2) = 1$
 - total error = 1

Based on total error, partition using $B \leq 1$. The resulting tree is in Figure 6. Since the error in partition $A \leq 1$, partition $A \leq 2$, and partition $B \leq 1$ is 0, only observations in the partition $B > 2$ (observations 8 and 10) are examined in the next iteration.

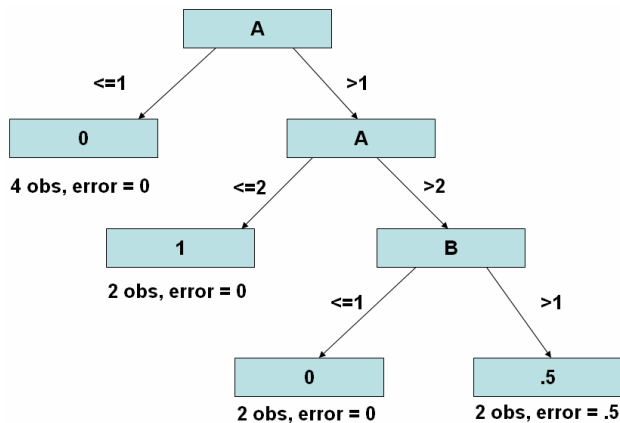


Figure 6. Tree after three iterations

Iteration 4

Minimum error or minimum observation reached for all partitions

To classify a new observation, the observation traverses the tree according to its predictors' values. Consider the following predictions:

Predictor values: $A = 3, B = 1$

Classification = 0 (not risky)

The path down the tree is shown in Figure 7.

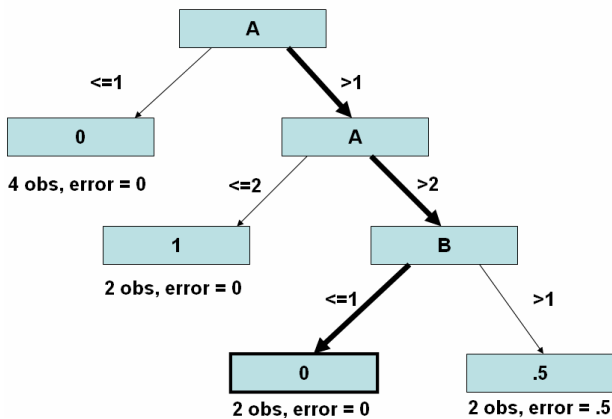


Figure 7. Path down classification tree

Predictor values: $A = 2, B = 2$

Classification = 1 (risky)

The path down the tree is shown in Figure 8.

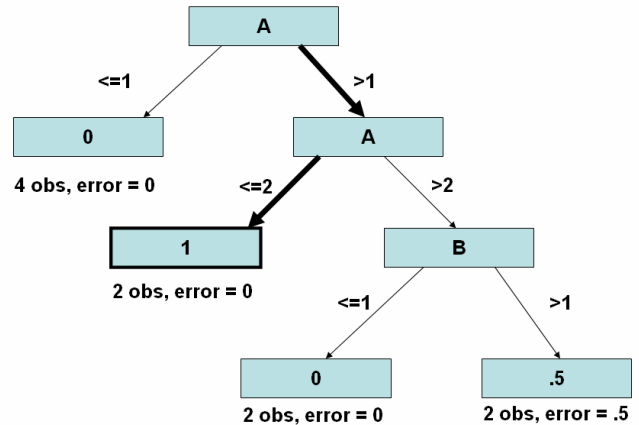
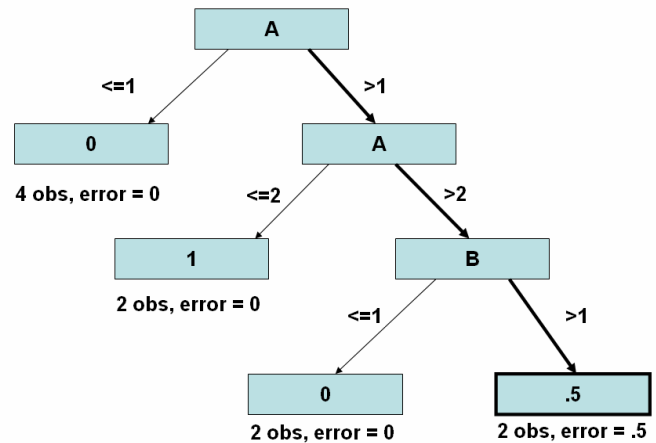


Figure 8. Path down classification tree

Predictor values: $A = 3, B = 2$

Classification = ? (unclear)



The prediction is unclear. In practice a cutoff is usually used to classify observations for partitions that are not homogenous such as in Khoshgootaar et. al. [40]. For example, if we use $>.75$ as the cut off, then the classification is 0 (not risky).

4.3.2 Techniques that produce level 1 output

Techniques that produce a level 1 output are concerned with putting observations into classes; therefore techniques that produce a level 1 output use the training set to determine how the predictors' values influence which class an observation belongs to. Different techniques determine the influence of prior observations differently or determine the class of a new observation differently.

Linear modeling (with model selection)

The logistic regression technique is the variant of the linear modeling technique that produces a level 1 output and is explained in detail by Weisburg in [104]. The idea behind linear modeling is that each increase in a predictor's value increases the probability that the observation belongs to one of the classes by the same amount.

The logistic regression technique involves fitting a parameterized linear model between the predictors' values and the logit transformed field problem metric value (e.g. 0 for risky and 1 for not risky). The parameter values are determined by minimizing a measure of the fit (such as residual sum of squares, absolute difference, least relative difference, etc.).

In most situations, linear modeling involves model section. Model selection fits models with sub-sets of predictors and selects a model that balances the bias-variance tradeoff. Model selection balances the trade-off by including only predictors that have the most amount of benefit or by dropping predictors that have the least amount of benefit as judged by a model selection criterion (e.g. AIC, BIC, Cross-validation).

Given a new observation, the predictors' values are inserted into the fitted linear model used to produce a real value between 0 and 1 representing the probability of the observation belonging to a class (e.g. risky). A pre-determined cutoff is used to classify the observation.

Research work that uses this technique includes: Briand et. al. [5] Jones et. al. [32], Khoshgoftaar et. al. [49], Mockus et. al. [74]

Trees (classification trees)

The trees technique involves creating partitions in the observations based on predictors' values that minimize the error in classifications within each partition and is explained in detail by Selby and Porter in [94]. The idea behind trees is that predictors have critical values that distinguish between classes; therefore by identifying the critical values, an observation can be classified using its predictors' values.

The partitioning process is repeated until the error within each partition is below some limit or until the number of observations within each partition is below some limit. The binary splitting process produces a tree. A predetermined cut off is usually used to assign a leaf to a class based on the proportion of observations in each class.

A new observation traverses the tree according to its predictors' values until the observation reaches a leaf node. The class of the leaf node is the predicted class of the new observation.

An example is given in section 4.3.1.

Research work that uses this technique includes: Briand et. al. [5], Ebert [13]/[14]/[15], Khoshgoftaar and Allen [37], Khoshgoftaar et. al. [40], Khoshgoftaar et. al. [41], Khoshgoftaar et. al. [43]/[44], Khoshgoftaar et. al. [50], Khoshgoftaar et. al. [51], Khoshgoftaar et. al. [53], Khoshgoftaar and Seliya [59], Khoshgoftaar et. al. [62]/[63], Khoshgoftaar et. al. [64], Kokol et. al. [66], Selby and Porter [94], Selby and Porter [95], Takahashi et. al. [96], Troster and Tian [99]

Discriminant analysis (with model selection)

The discriminant analysis technique involves dividing observations in the training set into classes (risky or not risky) and then when a new observation needs to be classified, the technique computes a closeness function to determine which class the new observation belongs to. The discriminant analysis technique is explained in detail by Khoshgoftaar et. al. in [45]. The idea behind discriminant analysis is that observations that belong to the same class share similarities in their predictors' values; therefore, a new observation's proximity to each class based on its predictors' values is used to determine the class of the new observation.

When a new observation, x , needs to be classified, a multi-variate probability density function, $f_k(x)$, is used to give the probability of the new observation being in each class, k . The probability density function is based on how close the predictors' values are to the predictors' values in the training set for each class. The probability of class membership and a pre-determined cut off (usually the prior proportion of observations in each class) are used to determine class membership. In most case, this technique also uses model section.

Research work that uses this technique includes: Karuthanithi [36], Khoshgoftaar and Allen [38], Khoshgoftaar et. al. [42], Khoshgoftaar et. al. [45]/[46], Khoshgoftaar et. al. [48], Khoshgoftaar et. al. [54]/[55], Kokol et. al. [66], Munson and Khoshgoftaar [75], Ohlsoon and Runeson [85], Pighin and Zamolo [89]

Rules

The rules technique captures rules of thumb and formally known relations among the facts. The rules are presented as if-then rules that associate a conclusion (i.e. a classification) with a set of antecedents. The rules technique is explained in detail by Yuan et. al. in [103]. The idea behind rules is that a set of if-then rules can decide which class an observation belongs to.

A new observation is classified by determining which rules apply to the new observation.

Research work that uses this technique includes: Ebert [13]/[14]/[15], Yuan et. al. [103]

Clustering

The clustering technique groups observations into clusters according to predictors' values and a distance function. The clustering technique is explained in detail by Khoshgoftaar et. al. in [56]. The idea behind clustering is that the predictors' values can be used to find similar observations (i.e. clusters) and that all members of the same cluster should belong to the same class.

A distance function specifies how close predictors' values need to be to the other members of a cluster to be included in a cluster. A majority function determines the class of a cluster based on the classes of the observations within the cluster.

A new observation is placed into one of the clusters based on a predictors' values. The class of the cluster is the predicted class of the new observation.

Research work that uses this technique includes: Khoshgoftaar et. al. [39], Khoshgoftaar et. al.[56], Yuan et. al. [103]

Neural networks

The neural networks technique simulates how a set of neurons or processing elements are interconnected through different connection strengths. The neural networks technique is explained in detail by Khoshgoftaar et. al. in [55]. The idea behind neural networks is that predictors' values are like neural inputs, which is used by the neural network to arrive at a conclusion about a new observation.

A neural networks model is a multi-layer perceptron model that produces a real value between 0 and 1, which indicates class membership. The predictors are in one layer, with each predictor as one neuron, and the output is in one layer. There is at least one intermediate hidden layer in between with different number of neurons. Each neuron in one layer is connected to each neuron in the next layer. The connection strength between the neurons can vary. A non-linear function is used to combine values coming into the neuron to produce the output from the neuron.

For a new observation, the predictors' values are placed on the outer layer and the predicted value between 0 and 1 is produced at the output neuron. A predetermined cut off is used to classify the observation.

Research work that uses this technique includes: Karuthanithi [36], Khoshgoftaar et. al. [42], Khoshgoftaar et. al. [54]/[55], Kokol et. al. [66], Xu et. al. [102]

Case based

The case based technique classifies a new observation by identifying similar cases and examining the classes of the similar cases. The case based technique is explained in detail by Khoshgoftaar et. al. in [39]. The idea behind case based is that similar cases can be used to determine the class of a new observation.

There is no training involved for case based models.

For a new observation, the case based technique determines training observations that are similar to the observation using predictors' values and a closeness function. Then, the class of the new observation is determined using the similar cases and a solution algorithm that determines class of the new observation based on the classes of the similar cases.

Research work that uses this technique includes: Khoshgoftaar et. al. [39], Schenker and Khoshgoftaar [91]

Sets

The sets technique ranks the predictors according to their ability to discriminate between classes, then it uses a subset to classify observations. The sets technique is explained in detail by Briand et. al. in [5]. The idea behind sets is that predictors' have critical values that distinguish between classes. This method is similar to the trees technique; however, the model construction process is not iterative.

The sets technique ranks the predictors according to their ability to discriminate between classes. The critical value that maximizes the difference between partitions is determined for each predictor. A Boolean function is then constructed using a subset of the predictors and their critical values to classify observations.

For a new observation, the Boolean function is applied to the predictors to derive the class of the new observation.

Research work that uses this technique includes: Briand et. al. [5], Khoshgoftaar and Seliya [61]

Linear programming

The linear programming technique involves cutting the n-dimensional space (representing the n predictors) using multi-dimensional planes. The linear programming technique is described in detail by Pighin et. al. in [90]. The idea is that predictors' values determine an observation's location in an n dimensional space and regions of the space (as defined by the planes) belong to the same class.

The cutting process is repeated until the homogeneity of each region is below a threshold or the number of observations in each region is below a threshold. A predetermined cut off is used to assign a class to each region based on the classes of the observations in the region.

A new observation is placed into one of the regions based on the predictor's value. The class of the region is the predicted class of the new observation.

Research work that uses this technique includes: Kokol et al.[66], Pighin et. al.[90]

Heuristics or any level 2 output with heuristic

The heuristics technique involves applying a heuristic rule (e.g. the Pareto distribution). The heuristics technique is explained in detail by Ebert in [13]/[14]/[15]. The idea behind heuristics is that a small percentage of observations account for most of the problems.

New observations are ranked according to a predictor's value or modeling output from a level 2 model, then a percentage of the observations are assigned to one class according to a heuristic.

Research work that uses this technique includes: Ebert [13]/[14]/[15], Kokol et. al.[66], Ohlsson and Wohlin [84], Ostrand et. al. [87]

4.3.3 Techniques that produce level 2 output

Techniques that produce a level 2 output are concerned with predicting a specific number; therefore techniques that produce a level 2 output use the training set to determine what the number of field problems will be given the predictors' values. Different techniques determine how the predictors' values influence the number of field problems differently.

Linear modeling (with model selection)

The linear regression technique and the negative binomial regression technique are the variants of the linear modeling technique that produces a level 2 output and is explained in detail by Weisburg in [104]. The idea behind linear modeling is that changes in a predictor's value changes the predicted number of field problems (or transformed form of field problems in the case of binomial modeling) by a fixed amount.

The transformation function for the negative binomial regression model is the log function. The fitting process is same as the linear modeling process to produce a level 1 output. Model selection technique is also usually used.

For a new observation, the predictors are inserted into the linear model to produce a prediction of the number of field problems.

Research work that uses this technique includes: Graves et. al. [22], Harter et. al. [24], Khoshgoftaar and Allen [38], Khoshgoftaar et. al. [39], Khoshgoftaar et. al. [47], Khoshgoftaar et. al. [52], Khoshgoftaar et. al.[54]/[55], Khoshgoftaar et. al.[56], Khoshgoftaar et. al. [57], Khoshgoftaar et. al. [58], Kokol et. al. [66], Mockus et. al. [74], Ostrand et. al. [87], Yuan et. al. [103]

Non-linear regression

The non-linear regression technique is similar to the linear modeling technique. It involves fitting a parameterized non linear model (e.g. a power function) between the predictors' values and the value of the field problem metric. The model fitting procedure is the same as the procedure for the linear modeling technique. The non-linear regression technique is explained in detail by Weisburg in [104]. The idea behind non-linear modeling is that a change in the predictor's value change the predicted number of field problems by a parameterized amount.

For a new observation, the predictors' values are inserted into the non-linear model to produce a prediction of the number of field problems.

Research work that uses this technique includes: Graves et. al. [22], Khoshgoftaar et. al.[52]

Trees (Regression trees)

This is the same technique used to produce a level 1 output except that the value of the field problem metric is predicted. Using the trees technique to produce a level 2 output is explained in detail by Khoshgoftaar and Seliya in [60]. The idea is that critical values identify similar observations and that all similar observations have similar numbers of field defects.

A new observation traverses the tree, then the mean or median of the values of the field problem metric in the leaf is taken as the predicted number of field problems for the new observation.

Research work that uses this technique includes: Khoshgoftaar and Seliya [60]

Neural networks

This is the same technique used to produce a level 1 output except that the output (a continuous value between 0 and 1) is scaled according to the range of the values of the field problem metric in the training set. Using the neural networks technique to produce a level 2 output is explained in detail by Khoshgoftaar et. al. in [57]. The idea behind neural networks is that predictors' values are like neural inputs and can be used by a neural network to arrive at a conclusion.

For a new observation, the predictors' values are used to produce a value between 0 and 1. Then the value is scaled up according to the range of the number of field problems in the training set.

Research work that uses this technique includes: Khoshgoftaar et. al. [57], Khoshgoftaar et. al. [58]

4.3.4 Principal component analysis (PCA)

The principal component analysis (also called singular value decomposition) technique produces a new set of

predictors using linear combinations of the original predictors and is explained in detail by Khoshgoftaar et. al. in [45]/[46].

The idea behind PCA is that there are only a few sources of true variation within a set of predictors and that many predictors are highly correlated with each other because they capture similar attributes. PCA solves this problem by constructing new predictors that capture the different sources of variation using linear combinations of the original predictors. The new predictors will be independent of each other and will contain all the information in the original predictors.

PCA tries to include all the variance captured in the original predictors while reducing the number of predictors. The new predictors (principal components) are in ranked order so that the first new predictor captures the most variation, the second predictor captures the second most, and so on. Usually, a subset of the principal components that capture a large proportion of the total variance (e.g. 90% as in Khoshgoftaar et. al. [45]) is then used.

Research work that uses this technique includes: Briand et. al. [5], Khoshgoftaar and Allen [38], Khoshgoftaar et. al. [42], Khoshgoftaar et. al. [45]/[46], Khoshgoftaar et. al. [47], Khoshgoftaar et. al. [48], Khoshgoftaar et. al. [56], Khoshgoftaar et. al. [64], Khoshgoftaar et. al. [62]/[63], Kokol et. al. [66], Munson and Khoshgoftaar [75], Ohlsson and Runeson [85], Pighin and Zamolo [89], Xu et. al. [102]

4.3.5 *Bagging, boosting, and logitboost*

Bagging, boosting, and logitboost are used by Khoshgoftaar et. al. in [53] to improve the predictions of individual models produced by the trees technique. The authors show that the accuracy of classifications can be improved by combining classifications from multiple models. The idea is that the training set used to build a model could be biased. By combining predictions from models built using different samples, a more accurate prediction can be made.

Bagging

The bagging technique randomly re-samples from the training set, fits a model for each re-sampled data set, and takes the consensus of the classifications as the output.

Boosting

The boosting technique is similar to the bagging techniques. However, it builds models that complement each other by building models that focus on data that previous models performed poorly on. In the boosting technique the re-sampling process is an iterative and weighted process, in contrast to the random process in the

bagging technique. Each time, the weight of correctly classified observations is decreased while the weight of misclassified instances is increased. Therefore, the model in the next iteration is more likely to focus on misclassified instances. In addition, the voting process is modified. The models that have better overall performance are given more weight in the voting process.

LogitBoost

The logitboost technique is a re-derivation of the AdaBoost as a method for fitting an additive model in a forward stepwise process. The idea is to fit an additive model by minimizing the squared loss in a forward stepwise manner.

4.3.6 *Fuzzy logic*

Fuzzy logic is used in systems where values can have degrees of truthfulness or falsehood represented by a range of values between 1 (true) and 0 (false) and is explained in detail by Schenker and Khoshgoftaar in [91]. The idea behind fuzzy logic is that information cannot always be described accurately (e.g. middle-aged: 40-50? 45-65?); therefore, the imprecision in information needs to be captured. Fuzzy logic describes the imprecision using intervals and probabilities. With fuzzy logic, the outcome of an operation can be expressed imprecisely and a probability distribution is assigned to values.

Research work that uses this technique includes: Ebert [13]/[14]/[15], Schenker and Khoshgoftaar [91], Xu et. al. [102]

4.3.7 *Combining techniques*

Each modeling method can comprises of several techniques. It is not clear what the complete set of valid combinations is. We discuss the combinations that have been explored in prior work. Research work that combines techniques and their findings are listed in Table 8.

Table 8. Research work with combination of techniques

Research work	Method	Type of output
Edbert [13]/[14]/[15]	Fuzzy logic and rules	Level 1
Khoshgoftaar and Allen [38] Khoshgoftaar et. al. [42] Khoshgoftaar et. al. [45]/[46] Khoshgoftaar et. al. [48] Khoshgoftaar et. al. [54]/[55] Kokol et. al. [66] Munson and Khoshgoftaar [75] Pighin and Zamolo [89]	Principal component analysis and discriminant analysis	Level 1
Briand et. al. [5]	Principal component analysis and logistic regression	Level 1
Schenker and Khoshgoftaar [91]	Fuzzy logic and case based	Level 1
Khoshgoftaar et. al. [53]	Bagging, boosting, and LogitBoost with trees	Level 1
Khoshgoftaar et. al. [62]/[63] Khoshgoftaar et. al. [64]	Principal component analysis and trees	Level 1
Khoshgoftaar et. al. [56]	Principal component	Level 2

	analysis, clustering, and linear modeling	
Xu et. al. [102]	Principal component analysis, fuzzy logic, and neural networks	Level 2
Yuan et. al.[103]	Fuzzy logic, clustering, and linear modeling	Level 2
Khoshgoftaar et. al. [47]	principal component analysis and linear regression	Level 2

4.3.7.1 Accuracy

The most widely used criterion for comparing modeling methods is accuracy; however, it is difficult to compare accuracy across research work due to differences such as different metrics, different modeling parameters, and environmental differences (e.g. organizational related differences). A few studies have compared predictions of different modeling methods in the same setting. Table 9 summarizes the findings. Based on the research work a partial ordering of methods using accuracy is in Figure 9.

Table 9. Findings of research work comparing accuracy of different modeling methods

Research work	Accuracy of preferred method	Accuracy of other methods
Briand et. al. [5]	<i>Sets</i> 7.81% type I error 4.11% type II error 6.04% overall error	<i>Linear modeling (logistic regression) with model selection</i> 23.44% type I error 32.88% type II error 28.47% overall error <i>Linear modeling (logistic regression) with principal component analysis and model selection</i> 20% type I error 28.77% type II error

		<p>24.64% overall error</p> <p><i>Classification trees</i></p> <p>16.67% type I error</p> <p>17.81% type II error</p> <p>7.24% overall error</p>
Ebert [13]/ [14] / [15]	<p><i>Fuzzy rules</i></p> <p>18.4% type I error</p> <p>21.6% type II error</p> <p>19% overall error</p>	<p><i>Heuristics</i></p> <p>10.43% type I error</p> <p>45.95% type II error</p> <p>18.5% overall error</p> <p><i>Trees</i></p> <p>8.59% type I error</p> <p>43.24% type II error</p> <p>15% overall error</p> <p><i>Discriminant analysis</i></p> <p>15.95% type I error</p> <p>32.43% type II error</p> <p>19% overall error</p>
Karuthanithi [36]	<p><i>Neural networks</i></p> <p><i>(trained using 25% of the data)</i></p> <p>20.19% type I error</p> <p>12.11% type II error</p> <p><i>(trained using 50% of the data)</i></p> <p>17.41% type I error</p> <p>15.04% type II error</p> <p><i>(trained using 90% of the data)</i></p> <p>9.77% type I error</p> <p>15.47% type II error</p>	<p><i>Discriminant analysis</i></p> <p><i>(trained using 25% of the data)</i></p> <p>13.16% type I error</p> <p>15.61% type II error</p> <p><i>(trained using 50% of the data)</i></p> <p>12.45% type I error</p> <p>16.01% type II error</p> <p><i>(trained using 90% of the data)</i></p> <p>14.17% type I error</p> <p>21.11% type II error</p>
Khoshgoftaar and Allen [38]	<p><i>Discriminant analysis with principal component analysis</i></p> <p>23.8% type I error</p> <p>13.7% type II error</p> <p>22.6% overall error</p>	<p><i>Discriminant analysis</i></p> <p>33.8% type I error</p> <p>16.3 % type II error</p> <p>31.7% overall error</p>
Khoshgoftaar et. al. [39]	<p><i>Case based</i></p> <p>16.0% type I error</p> <p>15.8% type II error</p> <p><i>Linear modeling (linear regression)</i></p> <p>16.0% type I error</p> <p>15.8% type II error</p>	<p><i>Clustering</i></p> <p>14.7% type I error</p> <p>21.1% type II error</p>
Khoshgoftaar et. al. [42]	<p><i>Neural networks</i></p>	<p><i>Discriminant analysis with principal component</i></p>

	26.0% type I error 26.9% type II error 26.2% over all error	<i>analysis and model selection</i> 27.9% type I error 39.4% type II error 29.5% overall error
Khoshgoftaar et. al. [54]/[55]	<i>Neural networks</i> 12.5% type I error 6.7% type II error 11% overall error	<i>Discriminant analysis with principal component analysis</i> 6.25% type I error 26.7% type II error 11% overall error
Khoshgoftaar et. al. [57]	<i>Neural networks</i> <i>(System 1)</i> .3980 ARE .28 standard deviation <i>(System 2)</i> .5467 ARE .08 standard deviation	<i>Linear modeling (linear regression) with model selection</i> <i>(System 1)</i> .5877 ARE .62 standard deviation <i>(System 2)</i> .9998 ARE 1.37 standard deviation
Khoshgoftaar et. al. [64].	<i>Trees with principal component analysis</i> <i>(release 2)</i> 29.3% type I 21.2% type II <i>(release 3)</i> 29.9% type I 19.1% type II <i>(release 4)</i> 32.7% type I 19.6% type II	<i>Trees</i> <i>(release 2)</i> 31.7% type I 23.3% type II <i>(release 3)</i> 30.3% type I 14.9% type II <i>(release 4)</i> 35.6% type I 22.8% type II
Kokol et. al. [66]	<i>Discriminant analysis with principal component analysis</i> 6.3% type I error 14.3% type II error 8.3% overall error	<i>Linear programming</i> 25.3% type I error 4.9% type II error 10.9% overall error <i>Trees</i> 15.1% type I error 22.2% type II error 17.0% overall error <i>Heuristics</i> 17.1% type I error 29.2% type II error 21.1% overall error

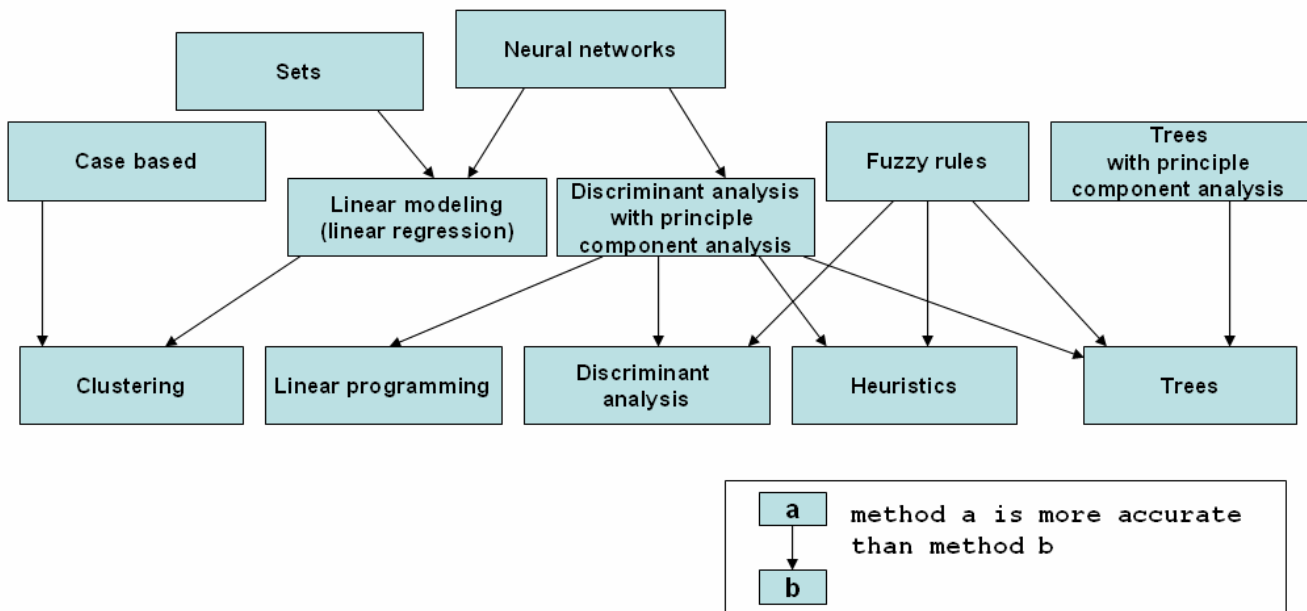


Table 9. Ordering of modeling methods using accuracy

4.3.8 Other methods of evaluation

The idea here is that in some situations the accuracy of predictions is not the most important criterion. Other methods of evaluating modeling methods have been proposed but are not frequently used. For example, a widely discussed criterion for comparing modeling methods is the explicability of the resulting model (i.e. how easy is it to interpret the effects of each predictor). This may be important if the objective of field problem prediction is to identify important predictors to plan for improvements.

To demonstrate explicability, consider the following two models produced by Khoshgoftaar et. al., one is a tree model from [60] and the other is a linear model using principal component analysis from [47]. Both models predict the number of field problems within a module.

RLSTOT: the number of vertices plus the number of arcs within loop control structure spans with a flow graph

NL: the number of loops with a flow graph

VG: Cyclomatic complexity

PCSTOT: the total number of arcs located within the span of conditional arcs in a flow graph

NELTOT: the total nesting level of all arcs

TCT: the number of calls to entry points

UCT: the number of unique entry points called by this module

IFTH: the number of arcs that contain a predicate of a control structure, but are not loops

NDI: the number of include files that this modules uses, including itself

ISNEW: if the module is new (1 for yes, 0 for no)
 ISCHG: if the module has been changed since last release (1 for yes, 0 for no)
 Tree model is in Figure 10:

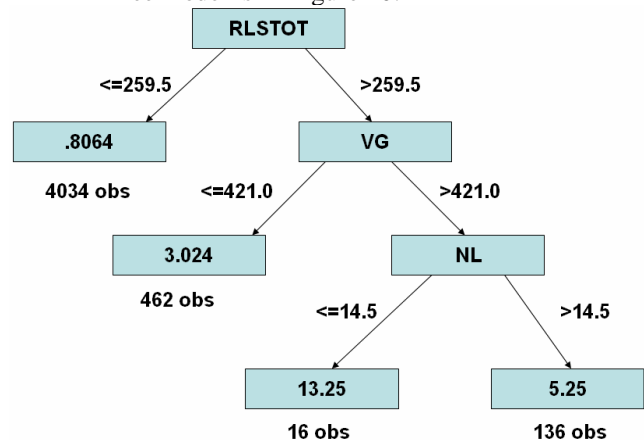


Figure 10. Classification tree from [60]

The principal components are in Table 10.

Table 10. principal components from [47]

Metric	Component 1	Component 2	Component 3
RLSTOT	.901	.359	.137
NL	.880	.370	.134
PCSTOT	.719	.545	.316
NELTOT	.683	.593	.334
TCT	.359	.864	.216
UCT	.426	.830	.245
VG	.597	.724	.309
IFTH	.599	.681	.357
NDI	.177	.265	.939

The linear model is:

$$\text{Field problems} = .520 + 1.233 (\text{ISCHG}) + .541 (\text{ISNEW}) + .577 (\text{Component 3}) + .368 (\text{Component 1}) + .338 (\text{Component 2})$$

The tree model is easily understood. The important predictors are clearly identified by internal nodes. Leaf nodes present the predicted number of field problems. The important values are clearly indicated.

The linear model with principal components analysis is not easy to understand due to the principal components. Components are constructed out of linear combinations of predictors. It is not clear what the contributions of each metric are. In addition, it is not clear which metrics are important.

Explicability is often discussed in literature, such as Ebert in [13]/[14]/[15], Khoshgoftaar et. al. in [42], and Khoshgoftaar et. al. in [45], but no established measure of explicability is used to compare modeling methods. There is no established measure of explicability since “easily understood” is a subjective measure and may differ from person to person.

5. CONCLUSION

We present the current state of research in metrics based models. Hopefully, this survey will help researchers who are interested in researching metrics based models and practitioners who wish to use metrics based models to predict field problems.

6. ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grand CCR-0086003, by the Sloan Software Industry Center at Carnegie Mellon University, and by the NASA High Dependability Computing Program under cooperative agreement NCC-2-1298.

7. REFERENCES

- [1] American Institute of Aeronautic and Astronautics. *Recommended practice for software reliability*. ANSI/AIAA 1993.
- [2] Victor Basili and Lionel Briand and Steven Condon and Yong-Mi Kim and Walcelio Melo and Jon Valett. Understanding and Predicting the Process of Software Maintenance Releases. In *Proceedings of ICSE*, 1996.
- [3] Victor Basili and Barry Perricone. Software Errors and Complexity: An Empirical Investigation. In *Communications of the ACM*, 1984.
- [4] Kathryn Bassin and P. Santhanam. Use of software triggers to evaluate software process effectiveness and capture customer usage profiles. In Eighth International Symposium on Software Reliability Engineering, 1997.
- [5] Lionel C. Briand and Victor R. Basili and Christopher J. Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. In *IEEE Transaction on Software Engineering*, 1993.
- [6] Sarah Brocklehurst and P.Y. Chan and Bev Littlewood and John Snell. Recalibrating Software Reliability Models. In *IEEE Transaction of Software Engineering*, 1990.
- [7] Sarah Brocklehurst and Bev Littlewood. New Ways to Get Accurate Reliability Measures. In *IEEE Software*, 1992.
- [8] Michael Buckley and Ram Chillarege. Discovering Relationships between Service and Customer Satisfaction. In *Proceedings of the International Conference on Software Maintenance 1995*.
- [9] Timothy A. Budd. Mutation analysis: ideas, examples, problems and prospects. In *Computer Program Testing*. (B. Chandrasekaran and S. Radicchi, editors), Elsevier North-Holland, 1981.
- [10] Sunita Chulani and P. Santhanam and Darrell Moore and Gary Davidson. Deriving a Software Quality View from Customer Satisfaction and Service Data. In *European Software Conference on Metrics and Measurement*, 2001.
- [11] Sunita Devnani-Chulani. Bayesian Analysis of Software Cost and Quality Models. In *Dissertation presented to the Faculty of the Graduate School University of Southern California*, 1999.
- [12] Edsger Wybe Dijkstra. Notes on structured programming. In *Structured Programming*, Academic press 1972.

- [13] Chistof Ebert. Evaluation and application of complexity-based criticality models. In *METRICS 1996*.
- [14] Chistof Ebert. Experiences with criticality predictions in software development. In *Proceedings of FSE 1997*.
- [15] Chistof Ebert. Industrial application of criticality predictions in software development. In *ISSRE 1998*.
- [16] Norman Fenton and Shari Pfleeger. *Software Metrics - A Rigorous and Practical Approach*. Chapman & Hall, London, 1997
- [17] Norman E. Fenton and Niclas Ohlsson. Quantitative Analysis of Faults and Failures in a Complex Software System. In *IEEE Transaction on Software Engineering*, 2000.
- [18] Norman Fenton and Martin Neil. Software metrics: road map. In *Proceedings of ICSE*, 2000.
- [19] Norman Fenton and William Marsh and Martin Neil and Patrick Cates and Simon Forey and Manesh Tailor. Making Resource Decisions for Software Projects. In *Proceedings of ICSE*, 2004.
- [20] Norman Fenton, R.W. Whitty, and AA Kaposi. A generalized mathematical theory of structured programming. In *Theoretical Computer Science*, Volume 36, 1985.
- [21] Lynn M. Foreman and Stuart H. Zweben. A study of the effectiveness of control and data flow testing strategies. In *J Systems Software*, 1993.
- [22] Todd L. Graves and Alan K. Karr and J.S. Marron and Harvey Siy. Predicting Fault Incidence Using Software Change History. In *IEEE Transaction on Software Engineering*, 2000.
- [23] Swapna Gokhale and W. Eric Wong and Kishor Trivedi and J. R. Hogan. An Analytical Approach to Architecture-Based Software Reliability Prediction. In *International Performance and Dependability Symposium*, 1998.
- [24] Maurice Halstead. *Elements of Software Science*. Elsevier, 1977.
- [25] Donald E. Harter and Mayuram S. Krishnan and Sandra A. Slaughter . Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. In *Management Science*, 2000.
- [26] William Hetzel. *The complete guide to software testing*. Collins, 1984.
- [27] IEEE standard for a software quality metrics methodology. In *IEEE Std 1061-1998*, 1998.
- [28] IEEE standard for software productivity metrics. In *IEEE Std 1045-1992*, 1993.
- [29] Z. Jelinski and Paul B. Moranda. Software reliability research. In *Statistical Methods for Evaluation of Computer Software Performance*, 1972.
- [30] Daniel.R. Jeske and M. Akber Qureshi. Estimating the failure rate of evolving software systems. In *11th International Symposium on Software Reliability Engineering*, 2000.
- [31] Capers Jones. *Applied software measurement, productivity and quality*, McGraw-Hill, 1996.
- [32] Wendell Jones and John P. Hudepohl and Taghi M. Khoshgoftaar and Edward B. Allen. Application of a Usage Profile in Software Quality Models. In *3rd European Conference on Software Maintenance and Reengineering*, 1999.
- [33] Garrison Kenney and Mladen A. Vouk. Measuring the Field Quality of Wide-Distribution Commercial Software. In *3rd International Symposium on Software Reliability Engineering*, 1992.
- [34] Garrison Kenney. Estimating Defects in Commercial Software During Operational Use. In *Transactions on Reliability*, 1993.
- [35] Garrison Kenney. The Next Release Effect in the Field Defect Model for Commercial Software. In *Thesis Submitted to the Graduate Faculty of North Carolina State University*, 1993.
- [36] M. Karuthanithi. Identifying fault-prone software modules using feed-forward networks: a case study. In *NIPS*, 1993.
- [37] Taghi M. Khoshgoftaar and Edward B. Allen. Predicting fault-prone software modules in embedded systems with classification trees. In *IEEE Symposium on High-Assurance Systems Engineering*, 1999.
- [38] Taghi M. Khoshgoftaar and Edward B. Allen. Multivariate assessment of complex software systems: a comparative study. In *IEEE International Conference on Engineering of Complex Computer Systems*, 1999.
- [39] Taghi M. Khoshgoftaar and Edward B. Allen and Jason C. Busboom. Modeling software quality: the software measurement analysis and reliability toolkit. In *IEEE Software*, 1996.
- [40] Taghi M. Khoshgoftaar and Edward B. Allen and Jianyu Deng. Using regression trees to classify fault-prone software modules. In *IEEE Transactions on reliability*, 2002.
- [41] Taghi M. Khoshgoftaar and Edward B. Allen and Jianyu Deng. Controlling over fitting in software quality models: experiments with regression trees and classification. In *METRICS*, 2001.

- [42] Taghi M. Khoshgoftaar and Edward B. Allen and John P. Hudepohl and Stephen J. Aud. Application of neural networks to software quality modeling of a very large telecommunications system. In *IEEE Transaction on Neural Networks*, 1997.
- [43] Taghi M. Khoshgoftaar and Edward B. Allen and Wendel Jones and John P. Hudepohl. Classification-tree models of software-quality over multiple releases. In *ISSRE*, 1999.
- [44] Taghi M. Khoshgoftaar and Edward B. Allen and Wendel Jones and John P. Hudepohl. Classification-tree models of software-quality over multiple releases. In *IEEE Transaction on Reliability*, 2000.
- [45] Taghi M. Khoshgoftaar and Edward B. Allen and Kalai S. Kalaichelvan and Nishith Goel. Early Quality Prediction: A Case Study in Telecommunications. In *IEEE Software*, 1996.
- [46] Taghi M. Khoshgoftaar and Edward B. Allen and Kalai S. Kalaichelvan and Nishith Goel. Predictive modeling of software quality for very large telecommunications systems. In *International Conference on Communications*, 1996.
- [47] Taghi M. Khoshgoftaar and Edward B. Allen and Kalai S. Kalaichelvan and Nitith Goel. Predictive modeling of software quality for very large telecommunications systems. In *IEEE International Conference on Communications*, 1996.
- [48] Taghi M. Khoshgoftaar and Edward B. Allen and Kalai S. Kalaichelvan and Nitith Goel and John Hedepohl and Jean Mayrand. Detection of fault-prone program modules in a very large telecommunications system. In *Proceedings of ISSRE*, 1995.
- [49] Taghi M. Khoshgoftaar and Edward B. Allen and Wendell D. Jones and John P. Hudepohl. Return on investment of software quality predictions. In *Workshop on Application-Specific Software Engineering*, 1998.
- [50] Taghi M. Khoshgoftaar and Edward B. Allen and Archana Naik and Wendell D. Jones and John P. Hudepohl. Using classification trees for software quality models: lessons learned. In *International High-Assurance Systems Engineering Symposium*, 1998.
- [51] Taghi M. Khoshgoftaar and Edward B. Allen and Xiaojing Yuan and Wendell D. Jones and John P. Hudepohl. Preparing measurements of legacy software for predicting operational faults. In *International Conference on Software Maintenance*, 1999.
- [52] Taghi M. Khoshgoftaar and Bibhuti B. Bhattacharyya and Gary D Richardson. Predicting software errors, during development, using nonlinear regression models: a comparative study. In *IEEE Transaction on reliability*, 1992.
- [53] Taghi M. Khoshgoftaar and Erik Geleyn and Laruent Nguyen. Empirical case studies of combining software quality classification models. In *Proceedings of QSIC*, 2003.
- [54] Taghi M. Khoshgoftaar and David L. Lanning and Abhijit S. Pandya. A comparative study of pattern recognition techniques for quality evaluation of telecommunications software. In *IEEE Journal on selected areas in communication*, 1994.
- [55] Taghi M. Khoshgoftaar and David L. Lanning and Abhijit S. Pandya. A neural network modeling methodology for the detection of high-risk programs. In *Proceedings of ISSRE*, 1993.
- [56] Taghi M. Khoshgoftaar and John C. Munson and David L. Lanning. A comparative study of predictive models for program changes during system testing and maintenance. In *Conference on software maintenance*, 1993.
- [57] Taghi M. Khoshgoftaar and Abhijit S. Pandya and David L. Lanning. Application of neural networks for predicting program faults. In *Annals of Software Engineering*, 1995.
- [58] Taghi M. Khoshgoftaar and Abhijit S. Pandya and Hermant B. More. A neural network approach for predicting software development faults. In *ISSRE*, 1992.
- [59] Taghi M. Khoshgoftaar and Naeem Seliya. Software quality classification modeling using the SPRINT decision tree algorithm. In *ICTAI*, 2002.
- [60] Taghi M. Khoshgoftaar and Naeem Seliya. Tree-based software quality estimation models for fault prediction. In *METRICS*, 2002.
- [61] Taghi M. Khoshgoftaar and Naeem Seliya. Improving usefulness of software quality classification models based on Boolean discriminant functions. In *Proceedings of ISSRE*, 2002.
- [62] Taghi M. Khoshgoftaar and Ruqun Shan and Edward B. Allen. Using product, process, and execution metrics to predict fault-prone software modules with classification trees. In *HASE*, 2000.
- [63] Taghi M. Khoshgoftaar and Ruqun Shan and Edward B. Allen. Improving tree-based models of software quality with principal component analysis. In *ISSRE*, 2000.

- [64] Taghi M. Khoshgoftaar and Vishal Thaker and Edward Allen. Modeling fault-prone modules of subsystems. In *Proceedings of ISSRE*, 2000.
- [65] Barbara Kitchenham and Shari Lawrence Pfleeger and Norman Fenton. Towards a framework for software measurement validation. In *IEEE Transaction on Software Engineering*, 1995.
- [66] P. Kokol and V. Podgorelec and M. Zorman and M Sprogar and M Pighin. An analysis of software correctness prediction methods. In *Second Asia-Pacific Conference on Quality Software*, 2001.
- [67] Jean-Claude Laprie. Dependability of computer systems: concepts, limits, improvements. In *ISSRE*, 1995.
- [68] Paul Luo Li, Mary Shaw, Kevin Stolarick, and Kurt Wallnau. The Potential for synergy between certification and insurance. In *Special edition of ACM SIGSOFT Int'l Workshop on Reuse Economics (in conjunction with ICSR-7)*, 2002.
- [69] Paul Luo Li and Mary Shaw and Jim Herbsleb and Bonnie Ray and P.Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. In *Proceedings of FSE*, 2004.
- [70] Paul Luo Li and Mary Shaw and Jim Herbsleb and Bonnie Ray and P.Santhanam. Empirical Evaluation of Defect Projection Models for Widely-deployed Production Software Systems. In *CMU tech report CMU-ISRI-04-130*, 2004.
- [71] Zhaohui Liu and Nalini Ravishanker and Bonnie Ray. Modeling Dynamic Reliability Growth Using Bayesian Methods. In *Reliability Review*, 2003.
- [72] Michael Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.
- [73] Thomas J. McCabe. A complexity measure. In *IEEE Transaction on Software Engineering*, 1976.
- [74] Audris Mockus and Ping Zhang and Paul Luo Li. Drivers for Customer Perceived Quality. In *Proceedings of ICSE*, 2005.
- [75] John C. Munson and Taghi M. Khoshgoftaar. The detection of fault-prone programs. In *IEEE Transactions on Software Engineering*, 1992.
- [76] John C. Munson and Taghi M. Khoshgoftaar. The dimensionality of program complexity. In *ICSE*, 1989.
- [77] John Musa and Anthony Iannino and Kazuhira Okumoto. *Software Reliability*. McGraw-Hill, 1990.
- [78] National Institute of Standards and Technology. *The economic impacts of inadequate infrastructure for software testing*. Planning Report 02-3, 2002.
- [79] Dinesh D. Narkhede. Bayesian Model for Software Reliability. www.ee.iitb.ac.in/uma/~dineshn/
- [80] Martin Neil and Norman Fenton. Predicting Software Quality Using Bayesian Belief Networks. In *Proceedings of 21st Annual Software Engineering Workshop NASA/Goddard Space Flight Centre*, 1996.
- [81] Martin Neil and Paul Krause and Norman Fenton. Software Measurement: Uncertainty and Casual Modeling. In *IEEE Software*, 2001.
- [82] Martin Neil and Paul Krause and Norman Fenton. A Probabilistic Model for Software Defect Prediction. In *IEEE Transaction in Software Engineering*, 2002.
- [83] Niclas Ohlsson and Hans Alberg. Predicting fault-prone software modules in telephone switches. In *IEEE Transactions on Software Engineering*, 1996
- [84] Magnus C. Ohlsson and Claes Wohlin. Identification of green, yellow, and red legacy components. In *ICSM*, 1998.
- [85] Magnus C. Ohlsson and Per Runeson. Experiences from replicating empirical studies on prediction models. In *METRICS*, 2002.
- [86] Thomas J. Ostrand and Elaine J. Weyuker. The Distribution of Faults in a Large Industrial Software System. In *Proceedings of ISSA*, 2002.
- [87] Thomas J. Ostrand and Elaine J. Weyuker and Thomas Robert M. Bell. Where the bugs are. In *Proceedings of ISSA*, 2004.
- [88] Maruizio Pighin and Anna Marzona. An empirical analysis of fault persistence through software releases. In *ISESE*, 2003.
- [89] Maruizio Pighin and Roberto Zamolo. A predictive metric based on discriminant statistical analysis. In *ICSE*, 1997.
- [90] Maruizio Pighin and Vili Podgorelec and Peter Kokol. Program risk definition via linear programming technique. In *METRICS*, 2002.
- [91] Donald F. Schenker and Taghi M. Khoshgoftaar. The application of fuzzy enhanced case-based reasoning for identifying fault-prone modules. In *International High Assurance Systems Engineering Symposium*, 1998.
- [92] Norman F. Schneidewind. Body of Knowledge for Software Quality Measurement. In *IEEE Computer*, 2002
- [93] Norman F. Schneidewind. Analysis of error processes in computer software. In *Sigplan Note*, 1975
- [94] Richard Selby and Adam Porter. Learning from examples: generation and evaluation of decision trees

- for software resource analysis. In *IEEE Transaction on Software Engineering*, 1988.
- [95] Richard Selby and Adam Porter. Software metric classification trees help guide the maintenance of large-scale systems. In *Proceedings of the Conference on Software Maintenance*, 1989.
- [96] Ryouei Takahashi and Yoichi Muraoka and Yurihiro Nakamura. Building software quality classification trees: approach, experimentation, evaluation. In *ISSRE*, 1997.
- [97] Jeff Tian. Integrating Time Domain and Input Domain Analyses of Software Reliability Using Tree-Based Models. In *IEEE Transaction on Software Engineering*, 1995
- [98] Joel Troster and Jeff Tian. Exploratory Analysis Tools for Tree-Based Models in Software Measurement and Analysis. In *Proceedings of SAST*, 1996.
- [99] Joel Troster and Jeff Tian. Measurement and Defect Modeling for a Legacy Software System. In *Annals of Software Engineering*, 1995.
- [100] W.N. Venables and Brian D. Ripley. *Modern Applied Statistics with S-plus, 4th edition*. Springer-Verlag, 2000
- [101] Michalis Xenos and Dimitris Stavrinoudis and Dimitris Christodoulakis. The Correlation Between Developer-oriented and User-oriented Software Quality Measurements (A Case Study). 5th European Conference on Software Quality, 1996.
- [102] Zhiwei Xu and Taghi M. Khoshgoftaar and Edward B. Allen. Prediction of software faults using fuzzy nonlinear regression modeling. In *HASE*, 2000.
- [103] Xiaohong Yuan and Taghi M. Khoshgoftaar and Edward B. Allen and K Gasesan. An application of fuzzy clustering to software quality prediction. In *IEEE Symposium on Application-Specific Systems and Software Engineering Technology*, 2000.
- [104] Sanford Weisburg. *Applied linear regression*. Wiley, 1985.
- [105] Lee J. White and Edward I. Cohen. A domain strategy for computer program testing. In *IEEE Transaction on Software Engineering*, 1980.
- [106] Alan Wood. Software reliability from the customer view. In *IEEE Computer*, 2003.
- [107] Elaine Weyuker. Evaluating software complexity measure. In *IEEE Transaction on Software Engineering*, 1988.
- [108] Hong Zhu and Patrick Hall and John May. Software Unit Test Coverage and Adequacy. In *ACM Computing Surveys*, 1997.