# 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 MODLES

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 discuses 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

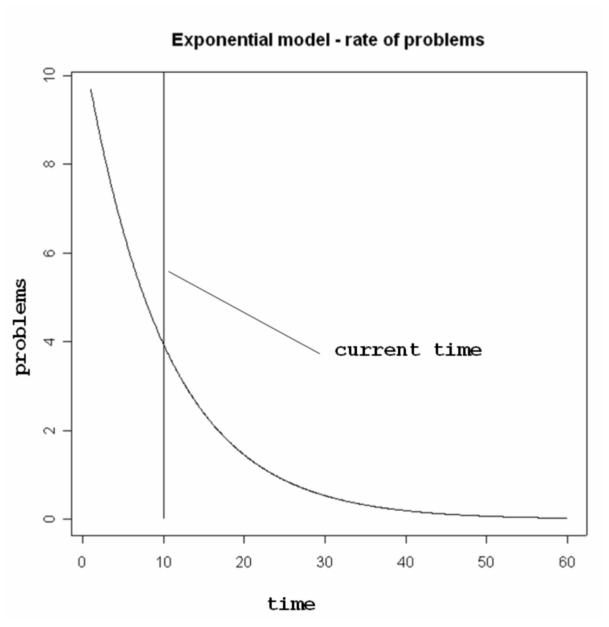


**Figure 2. Failure intensity function for the exponential model**

#### *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 McDonell 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 Khoshgotaar 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 Khoshgotaar 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 Khoshgotaar 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 the 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 is 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 software 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 | | |
| | $\beta_0$ | 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 | | |
| | $\beta_2$ | .234 |
| | s.e | .108 |
| | T | 2.160 |
| | P | 0.020 |
| ln(Product-Design-Complexity) subjective evaluation | | |
| | $\beta_3$ | -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.

FILINCUQ: 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

Logit(Faults) = -5.13 + .0284 FILINCUQ + .0209 LGPATH + .00043 VARSPMAX
Each predictor is significant at the 15% level

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

Logit(faults) = -5.13 + .0284 FILINCUQ + .0209 LGPATH + 1.2718 USAGE + .00043 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<br>18.75% type II error<br>25.2% overall error | 24.8% type I error<br>15.0% type II error<br>23.6% overall error |
| Khoshgoftaar et. al. [45] | Level 1 | Development | Product | 32.4% type I error<br>21.3% type II error<br>31.1% overall error | 23.8% type I error<br>13.8% type II error<br>22.6% overall error |
| Khoshgoftaar et. al.[50] | Level 1 | Development | Product | 27.0% type I error<br>27.4% type II error | 26.2% type I error<br>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<br>34.24% type II error | 29.06% type I error<br>30.77% type II error |
| Khoshgoftaar et. al. [51] | Level 1 | Deployment and usage | Product<br>Development | 23.55% type I error<br>32.80% type II error | 30.30% type I error<br>23.81% type II error |
| Khoshgoftaar et. al. [64] | Level 1<br>over multiple releases<br>(release 2) | Development | Product<br>Deployment and usage | 26.6% type I error<br>24.9% type II error | 29.3% type I error<br>21.2% type II error |
| Khoshgoftaar et. al. [64] | Level 1<br>over multiple releases<br>(release 3) | Development | Product<br>Deployment and usage | 28.8% type I error<br>21.3% type II error | 29.9% type I error<br>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]<br>Bassin and Santhanam [4]<br>Fenton and Ohlsson [17]<br>Harter et. al. [24]<br>Ohlsson and Wohlin [84]<br>Ostrand and Weyuker [86]<br>Pighin and Marzona [88]<br>Troster and Tian [99] |
| Level 1 | A categorical output | Is it risky or not? (Is the number of field defects above a threshold?) | Briand et. al.[5]<br>Ebert [13]/[14]/[15]<br>Jones et. al. [32]<br>Karuthanithi [36]<br>Khoshgoftaar and Allen [37]<br>Khoshgoftaar and Allen [38]<br>Khoshgoftaar et. al. [39]<br>Khoshgoftaar et. al. [40]<br>Khoshgoftaar et. al. [41]<br>Khoshgoftaar et. al. [42]<br>Khoshgoftaar et. al. [43]/[44]<br>Khoshgoftaar et. al. [45]<br>Khoshgoftaar et. al. [48]<br>Khoshgoftaar et. al. [49]<br>Khoshgoftaar et. al. [50]<br>Khoshgoftaar et. al. [51]<br>Khoshgoftaar et. al. [53]<br>Khoshgoftaar et. al. [54]/[55]<br>Khoshgoftaar and Seliya |

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

#### *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 o 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 result in a 33.33% type I error. One observation is classified as not risky when it 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<br>66.67% | 2<br>33.33% | 6 |
| **Actual risky** | 1<br>25% | 3<br>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 Khosghgoftaar 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

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

Absolute relative error is defined by Khosghgoftaar 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.

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

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: $\Sigma_{\text{partitions}} \Sigma_{\text{all observations in partition}} |y_i - \tilde{y}|$

$\tilde{y}$ = 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.

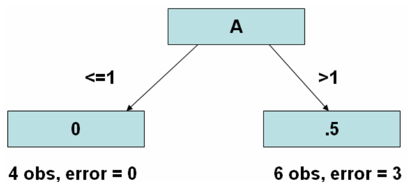


**Figure 4. Tree after one iteration**

*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.

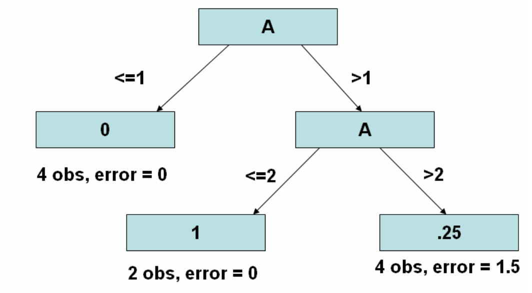


**Figure 5. Tree after two iterations**

*Iteration 3*

Minimum error or minimum observation not reached for all partitions.

Generate candidate partitions using predictors' values:

- A <=1: not possible

- A <=2: not possible
- B <= 1
  - error in partition 1 (B<=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<=1. The resulting tree is in Figure 6. Since the error in partition A<=1, partition A<=2, and partition B<=1 is 0, only observations in the partition B>2 (observations 8 and 10) are examined in the next iteration.

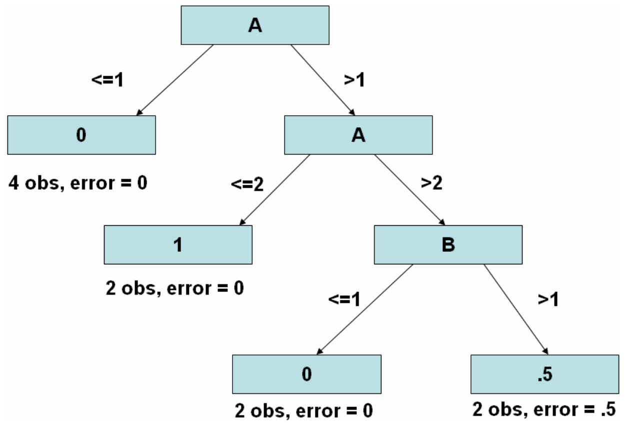


**Figure 6. Tree after three iterations**

*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.

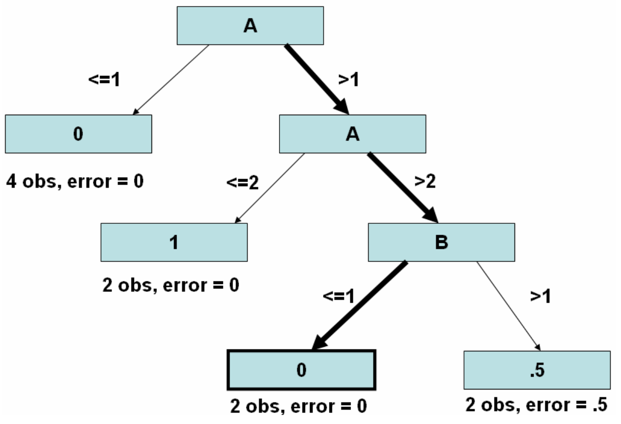


**Figure 7. Path down classification tree**

Predictor values: A= 2, B =2

Classification =1 (risky)

The path down the tree is shown in Figure 8.

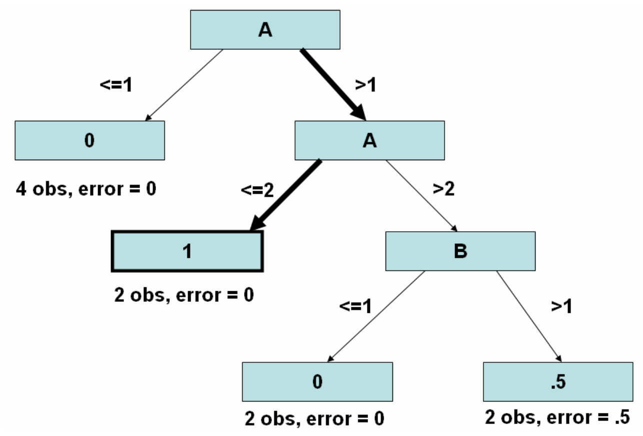
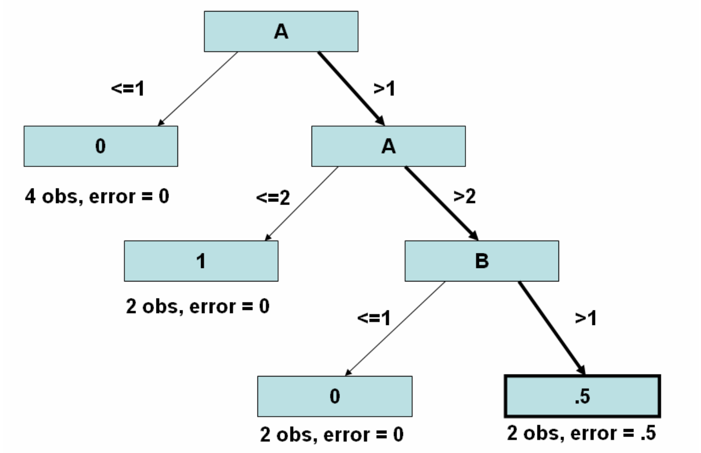


**Figure 8. Path down classification tree**

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 Khoshgotaar 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 Khoshgotaar 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]<br>Khoshgoftaar et. al. [42]<br>Khoshgoftaar et. al. [45]/[46]<br>Khoshgoftaar et. al. [48]<br>Khoshgoftaar et. al. [54]/[55]<br>Kokol et. al. [66]<br>Munson and Khoshgoftaar [75]<br>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]<br>Khoshgoftaar et. al. [64] | Principal component analysis and trees | Level 1 |
| Khoshgoftaar et. al. [56] | Principal component analysis, clustering, and linear modeling | Level 2 |
| 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] | *Sets*<br>7.81% type I error<br>4.11% type II error<br>6.04% overall error | *Linear modeling (logistic regression) with model selection*<br>23.44% type I error<br>32.88% type II error<br>28.47% overall error<br>*Linear modeling (logistic regression) with principal component analysis and model selection*<br>20% type I error<br>28.77% type II error |

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

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

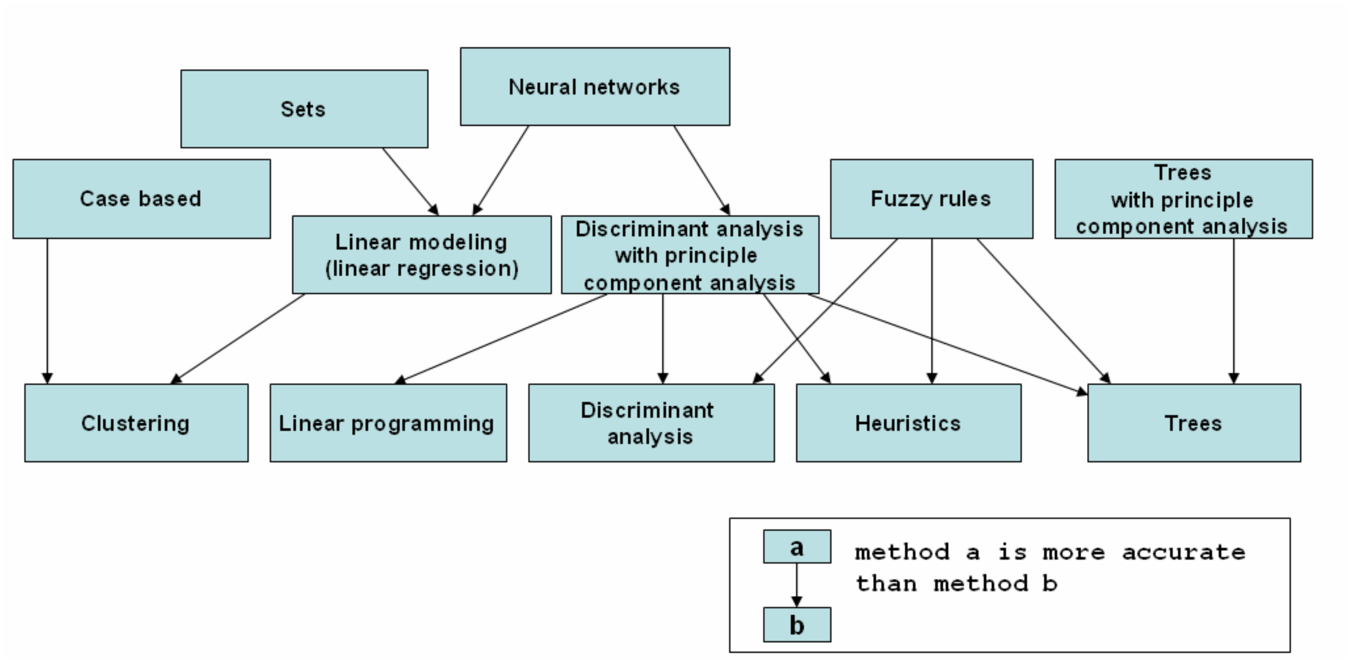

Sets
Neural networks
Case based
Linear modeling (linear regression)
Discriminant analysis with principle component analysis
Fuzzy rules
Trees with principle component analysis
Clustering
Linear programming
Discriminant analysis
Heuristics
Trees
a
b
method a is more accurate than method b

**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:

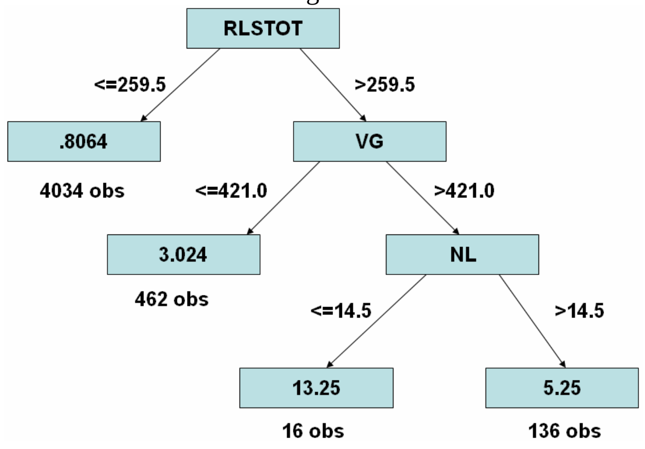


**Figure 10. Classification tree from [60]**

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:

Field problems = .520 + 1.233 (ISCHG) + .541 (ISNEW) + .577 (Component 3) + .368 (Component 1) + .338 (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. SUMMARY RESEARCH WORK

Using the classification schemes in section 4, we review research work in metrics based models. The summaries are ordered by the output level and year of publication. Each summary give a short overview of the study, the metrics used, and the results.

### 5.1 What predicts field problems?

This section reviews research work that addresses the question what predicts field problems. The studies are summarized in Table 11.

**Table 11. Summary of research work that address what predicts field problems**

| **Research overview** | **Predictors and field problem metrics** | **Results** |
|---|---|---|
| Basili and Perricone establish relationships between development metrics and field problems and between content metrics and field problems using the trends technique in 1984 [3].<br><br>The authors establish what predicts the field problems. The study uses data from multiple releases of a general-purpose program for satellite planning studies at the Software Engineering Laboratory. | The predictors are: reuse status, lines of code, and Cyclomatic complexity.<br><br>The measure of field problems is defect density per thousand lines of code. | Larger modules have lower defect densities.<br><br>Modules with higher Cyclomatic complexity have lower defect densities. |
| Troster and Tian establish a relationship between content metrics, development metrics, and field problems using the trees technique in 1995 [99].<br><br>The authors establish what predicts field problems. The study uses data from one release of an IBM relational database management system. | The authors use predictors classified into four classes:<br>Design:<br>high level structural complexity, high level data complexity, high level system complexity (calls to other modules and I/O variables), module level structural complexity, module level data complexity, module level system complexity<br>Size: | The presence of APARs are related to the number of referenced declarations, software maturity index, unreferenced declarations, changed source instruction, and high-level system complexity |

| | | |
|---|---|---|
| | lines of code, comment lines<br>Change:<br>changed source instructions, software maturity index<br>Complexity:<br>Cyclomatic complexity, referenced declarations, unreferenced declarations, statement count, procedure count<br>The measure of field problems is APARs resulting in fixes (Authorized Program Analysis Report). | |
| Bassin and Santhanam establish relationships between development metrics and field problems and between hardware and software configurations metrics and field problems using the trends technique in 1997 [4].<br><br>The authors establish what predicts field problems. The study uses data from multiple releases of a mainframe product and from multiple releases of a workstation product at IBM. | The predictor is development PTRs (problem tracking reports).<br><br>The measure of field problems is APARs. | Proportion of different types of development problems are related to the proportion of different types of APARs. |
| Ohlsson and Wohlin establish a relationship between content metrics and field problems using trends in 1998 [84].<br><br>The authors establish what predicts field problems. The study uses data on 28 software components from two releases of a real-time telecommunications software system. | The predictors are: number of output symbols in design doc, number of input symbols in design doc, number of unique external outputs from a components, number of unique external inputs to a component, number of state symbols in design doc, Cyclomatic complexity adapted for design doc, number of design pages in design doc, number of task symbols in the design doc, number of if boxes in the design doc.<br><br>The measure of field problems is fault reports from test and operation. | Cyclomatic complexity and states symbols are good indicators problematic modules. |
| Fenton and Ohlsson establish relationships between development metrics and field problems and between content metrics and field problems using trends in 2000 [16].<br><br>The authors establish what predicts field problems. The study uses data from two releases of a telecommunications switching system. | The predictors are: lines of code, Cyclomatic complexity, development defects, and modified signals.<br><br>The measure of field problems is defects found during the first year of development. | Post-release defects tend to occur in modules with no pre-release defects.<br>Lines of code is good at ranking modules with defects.<br>The combination of Cyclomatic complexity and modified signals are good at ranking modules with defects. |

| | | |
|---|---|---|
| Harter, Krishnan, and Slaughter establish a relationship between content metrics, development metrics, and field problems using the linear modeling technique in 2000 [24].<br><br>The authors establish what predicts field problems. The study uses data from 30 software products created by the systems integration division of an IT firm. | The predictors are: CMM levels, lines of code, and a subjective measure of product, and design complexity.<br><br>The measure of field problems is defects. | Higher process maturity decreases the number of defects.<br>More lines of code decrease the number of defects.<br>Higher product and design complexity increase number of defects. |
| Ostrand and Wyuker establish relationships between content metrics and field problems and between development metrics and field problems using the trends technique in 2002 [86].<br><br>The authors establish what predicts the number of field problems. The study uses data from multiple releases of an inventory tracking system at AT&T. | The predictors are: fault density per thousand lines of code, lines of code, reuse status, and development faults.<br><br>The measure of field problems is faults. | Larger files have lower fault densities.<br>Post-release faults tend to occur in files with no pre-release faults.<br>New files have both more faults and a higher fault density than pre-existing files. |
| Pighin and Marzona establish a relationship between content metrics and field problems and development metrics and field problems using trends in 2003 [88].<br><br>The authors establish what predicts field problems. The study uses data from 23 release of a management application and 15 releases of a medical application. | The predictors are: reuse status, Cyclomatic complexity, and lines of code.<br><br>The measure of field problems are faults and fault density. | More faults in older files but that it is due to the increasing number of old files.<br>Fault density is similar between new files and old files.<br>Approximately 50% of the faults are in files taking 35% of the lines of code. |

## 5.2 Is it risky or not?

This section reviews research work that addresses the question is it risky or not. The studies are summarized in Table 12.

**Table 12. Summary of research work that address is it risky or not**

| **Research overview** | **Predictors and field problem metric** | **Modeling specific details** | **Results** |
|---|---|---|---|
| Selby and Porter establish a relationship between content metrics, development metrics, and field | The authors use predictors classified into two classes:<br>Development effort attributes: design effort*, code effort, design effort per module call*, design | The authors classify modules as risky (number of faults in the top quantile of all modules) or not risky for modules in the next project. | The authors observe 0% combined type I and type II error using all metrics.<br><br>The authors observe a 30.5% |

|  |  |  |  |
|---|---|---|---|
| problems using the trees technique in 1988 [94].<br><br>The authors classify modules as risky or not risky. The study uses data from 16 project covering 4700 modules in a ground support software system at NASA. | effort per code effort, design effort per comment*, design effort per function call*, design effort per function plus module call*, design effort per input-output statement*, design effort per input-output parameter*, overhead effort, percent code effort of total development effort, percent design effort of total development effort, test effort, total effort, total development effort per 1000 source lines, total development effort per 1000 executable statements<br><br>Design and implementation style attributes:<br>assignment statement per 1000 executable statements, module calls per comment*, module calls per function plus module call*, module calls per input-output parameter*, comments*, function calls per module call*, function calls per comment*, function calls per function plus module call*, function calls per input-output parameter*, function plus module calls per comment*, function plus module calls per input-output statement*, function plus module calls per input-output parameters*, function plus module calls per 1000 source lines, function plus module calls per 1000 executable statements, function plus module calls*, input-output statements per comment*, input-output statements per input-output parameter*, input-output statements per 1000 executable statements, input-output parameters per comment*, source lines , source lines minus comments, Cyclomatic complexity, Cyclomatic complexity per 1000 source lines, Cyclomatic complexity per 1000 executable statements, assignment statements, module | The modules from 13-15 projects are used to train the module | combined type I and type II error using metrics available early in the development process. |

| | | | |
|---|---|---|---|
| | calls*, decisions, format statements, function calls*, input-output statements*, input-output parameters*, operands, operators, origin*, versions, execuTable statements, total operands, total operators. The metrics marked with * are available early in the development process.<br><br>The measures of field problems are faults. | | |
| Selby and Porter establish a relationship between content metrics, development metrics, and field problems using the trees technique in 1989 [95].<br><br>The authors classify modules as risky or not risky. The authors represent results in [94]. In addition, the study uses data on 907 files from a Hughes system collected over 54 months. | The authors no not reveal the data source or the predictors used.<br><br>The measures of field problems are faults. | The authors classify modules as risky (number of faults in the top quantile of all modules in the project) or not risky. | The authors observe an 18.84% type I error and a 24.32% type II error. |
| Munson and Khoshgoftaar establish a relationship between content metrics and field problems using the discriminant analysis technique with principal component analysis and model selection in 1992 [75].<br><br>The authors classify modules as risky or not risky. The study uses data on 327 modules from one | For the military system, the predictors: number of unique operators, number of unique operands, number of total operators, number of total operands, program length, program volume, Halstead's effort metric, Cyclomatic complexity, extended Cyclomatic complexity, number of procedure calls, number of comment lines, number of blank lines, number of lines of code, and number of executable lines of code.<br><br>The measure of field problems is faults. | The authors use principal component analysis to combine the predictors into principal components and choose the first two components for both systems. The authors divide data into a training set and a testing set.<br><br>The medical imaging system has 390 programs available for analysis. 260 programs are used to for training and 130 are used for testing.<br><br>The authors identify three groups in the training set: not risky (1 or 0 code changes) containing 126 observations, risky (10 or more | The authors observed a 10% type I error, a 13% type II error, and a 12% overall error classifying observations in the testing set that are risky and not risky (discarding 62 observations).<br><br>The authors redefine risky (more than 5 changes) and not risky (0-5 changes), and use a .80 classifying threshold to classify observations, which allows 110 out of 130 observations to be classified. The authors observe a 10% type I error, a 7% type II error, and a 17% overall error. |

| | | | |
|---|---|---|---|
| release of a military telecommunications software system and data on 390 programs one release of a medical imaging system. | For the medical system, the predictors are: lines of code including comments, lines of code, total character count, tot comments, number of comment characters, number of code characters, Halstead's program length, Halstead's estimated program length, Jenson's estimator of program length, Cyclomatic complexity, and Belady's bandwidth metric.<br><br>The measure of field problems is the number of fault related to code changes. | code changes) containing 30 observations, and other (2-9 code changes) containing 234 observations.<br>The authors only use observations that are risk and not risky to train the model.<br><br>The military communications system has 327 modules available for analysis. 218 programs are used to for training and 109 are used for testing.<br>The authors identify three groups in the training set: not risky (0 faults) containing 75 observations, risky (5 or faults) containing 59 observations, and other (1-5 faults) containing 84 observations.<br>The authors only use observations that are risk and not risky to train the model. | The authors observed a 3% type I error, a 26% type II error, and an 8% overall error classifying observations in the testing set that are risky and not risky (discarding 31 observations).<br>The authors redefine risky (more than 5 faults) and not risky (0-5 faults), and use a .80 classifying threshold to classify observations, which allows 86 out of 109 observations to be classified. The authors observe a 3% type I error, a 2% type II error, and a 5% overall error. |
| Briand, Basili, and Hetmanski establish a relationship between content metrics and field problems using the sets technique, the trees technique, the linear modeling (logistic regression) technique with model selection, and the linear modeling (logistic regression) technique with principal component analysis and model selection in 1993 [5].<br><br>The authors classify modules as risky or not risky. The study uses data on 146 components from an Ada system with the aid of an automated tool. | The predictors are: total number of cascaded program unit declarations/maximum possible number of cascaded program unit declarations, cascade imported program unit declarations/direct imported program unit declarations, number of library unit aggregations that contain a with statement to this compilation unit, number of compilation units that contain a with statement to this compilation unit, number of parameters per program unit declaration, fraction of old (reused verbatim) number of components, fraction of old (reused verbatim) number of SLOC, number of ADA statements, unique imported declarations/unique exported declarations, maximum statements nesting level, average statement nesting level, source lines of code, Halstead's volume, Cyclomatic complexity, number of declared variables. | The authors predict if a unit will be risky (at least one fault) or not risky (no faults). The authors use a cross-validation technique to evaluate models. | The authors observe a 20% type I error, a 28.77% type II error, and a 24.64% overall error for logistic regression with principal components and model selection.<br>The authors observe a 23.44% type I error, a 32.88% type II error, and a 28.47% overall error for logistic regression with model selection.<br>The authors observe a 16.67% type I error, a 17.81% type II error, and 17.24% overall error for classification trees.<br>The authors observe a 7.81% type I error, a 4.11% type II error, and a 6.04% overall error for optimized set reduction. |

| | | | |
|---|---|---|---|
| | The measure of field problems is faults. | | |
| Karunanithi establish a relationship between content metrics and field problems using the discriminant analysis technique and the neural network technique in 1993 [36].<br><br>The authors classify modules as risky or not risky. The study uses data from one release of a medical imaging system. The data is the same as in [75]. | The authors use the predictors: lines of code including comments, lines of code, total character count, total comments, number of comment characters, number of code characters, Halstead's program length, Halstead's estimated program length, Jenson's estimator of program length, Cyclomatic complexity, and Belady's bandwidth metric.<br><br>The measure of field problems is the number of fault related to code changes. | The authors then divide the data into training and testing sets of different sizes, training using 25% (testing with 75%) 33%, 50%, 67%, 75%, and 90% of the data.<br><br>The authors identify modules as not risky (1 or 0 code changes) or risky (10 or more code changes), and other (2-9 code changes). The authors do not consider the 'other' group.<br><br>The authors build a discriminant analysis model, a one layer neural network model (perceptron model) and a multi-layer neural network model. | Of the remaining modules, the authors observed a 13.16% type I error and a 15.61% type II error for the discriminant analysis technique trained using 25% of the data.<br><br>The authors observed a 12.45% type I error and a 16.01% type II error for the discriminant analysis technique trained using 50% of the data.<br><br>The authors observed a 14.17% type I error and a 21.11% type II error for the discriminant analysis technique trained using 90% of the data.<br><br>The authors observed a 16.17% type I error and a 15.98% type II error for the perceptron technique trained using 25% of the data.<br><br>The authors observed a 11.58% type I error and a 16.97% type II error for the perceptron technique trained using 50% of the data.<br><br>The authors observed a 4.03% type I error and a 19.11% type II error for the perceptron technique trained using 90% of the data.<br><br>The authors observed a 20.19% type I error and a 12.11% type II error for the neural networks technique trained using 25% of the data.<br><br>The authors observed a 17.41% type I error and a 15.04% type II error for the neural networks technique trained using 50% of the data.<br><br>The authors observed a 9.77% type I error and a 15.47% type II error for the neural networks technique trained using 90% of |

| | | | |
|---|---|---|---|
| | | | the data. |
| Khoshgoftaar, Lanning, and Pandya establish a relationship between content metrics and field problems using the discriminant analysis technique with principal component analysis of variables and the neural networks technique in 1994 [54]/[55].<br><br>The authors classify modules as risky or not risky. The study uses data from 282 modules in one release of a military telecommunications software system. | The predictors are: number of unique operators, number of unique operands, number of total operators, number of total operands, lines of code, executable lines of code, McCabe's Cyclomatic complexity, and extended Cyclomatic complexity.<br><br>The measure of field problems is the number of faults. | The authors use principal component analysis and select the first two components.<br>The authors construct a model using the neural networks technique using all predictors.<br><br>The authors divide data into a training set and a testing set. The training set has 188 observations and the testing set has 94 observations.<br><br>The authors divide the data into three classes: not risky (0 faults) containing 93 observations, risky (5 or more faults) containing 28 observations, or other (1-4 faults) containing 67 observation. The authors only use data in the risky and not risky categories to train models.<br><br>The authors classify risky modules (15 observations) and not risky modules (48 observations). | The authors observed a 6.25% type I error, a 26.7% type II error, and an 11% overall error using the discriminant analysis model.<br><br>The authors observed a 12.5% type I error, a 6.7% type II error, and an 11% overall error using the neural networks model. |
| Khoshgoftaar, Allen, Kalaivhelvan, Goel, Hudepohl, and Mayrand establish a relationship between content metrics, development metrics, and field problems using the discriminant analysis technique with principal component analysis and variable selection in 1995 [48].<br><br>The authors classify modules as risky or not risky. The study uses data on 1980 modules from a telecommunications | The predictors are: breaches of structure, conditional arc complexity, commented control structures, control statements, declaration statements, executable statements, non-loop conditional arc, lines of code, loops, modules used, nesting level, pending vertices, statements in loops, span of conditional arcs, vertices + arcs within loop spans, total calls to other modules, unique calls to other modules, volume of comments in declarations, volume of comments in structures, Cyclomatic complexity, module is new, and module is modified.<br><br>The measure of field problems is faults. | The authors use principal component analysis to construct principal components out of the content metrics then use variable selection to select four principal components, if a model is new, and if a model is modified as predictors.<br><br>The authors use 1230 observations to train the models and test the models using 660 observations. The authors classify modules as not risky (0-4 faults) and risky (5 or more faults). | The authors observe a 24.8% type I error, a 15.0% type II error, and a 23.6% overall error.<br><br>The authors also fit a module with out development information by not including the module is new and module is modified indicator variables. They observed a 26.0% type I error, an 18.75% type II error, and a 25.2% overall error. |

| | | | |
|---|---|---|---|
| system. | | | |
| Khoshgoftaar and Allen establish a relationship between content metrics, development metrics, and field problems using the linear programming technique, the discriminant analysis technique with model selection, and the discriminant analysis technique with principal component analysis and model selection in 1995 [38].<br><br>The authors classify modules as risky or not risky. The study uses data on 1980 modules from a telecommunications system. | The authors use predictors grouped in three classes:<br>Call graph metrics:<br>modules used, total calls to other modules, unique calls to other modules<br><br>Control flow graph metrics:<br>if-then conditional arcs, loops, nesting level, span of conditional arcs, span of loops, and McCabe Cyclomatic complexity.<br><br>Reuse status:<br>if a module is new or old, if a module is unchanged or modified<br><br>The measure of field problems is faults. | The authors use 1320 modules as the training set and 660 modules as the testing set. The authors classify modules as risky (1-4 faults) and not risky (5 or more faults). | The authors use only the metric ‘modules used’ as the predictor and observe a 1.4% type I error, an 80.0 type II error, and a 10.9% over all error.<br><br>The authors use the discriminant analysis technique with model selection construct a model and observe a 33.8% type I error, a 16.3% type II error, and a 31.7% overall error.<br><br>The authors use the discriminant analysis technique with principal component analysis and model selection and observe a 23.8% type I error, a 13.7% type II error, and a 22.6% overall error. |
| Khoshgoftaar, Allen, Kalaichelvan, and Goel establish a relationship between content metrics, development metrics, and field problems using the discriminant analysis technique with principal component analysis and variable selection in 1996 [45].<br><br>The authors classify modules as risky or not risky. The study uses data on 1980 modules from a telecommunications system. | The authors use predictors grouped in three classes:<br>Call graph metrics:<br>modules used, total calls to other modules, unique calls to other modules<br><br>Control flow graph metrics:<br>if-then conditional arcs, loops, nesting level, span of conditional arcs, span of loops, and McCabe Cyclomatic complexity.<br><br>Reuse status:<br>if a module is new or old, if a module is unchanged or modified<br><br>The measure of field problems is faults. | The authors use principal component analysis to construct principal components out of the content metrics then use variable selection to select three principal components, if a model is new, and if a model is modified as predictors.<br><br>The authors use 1230 observations to train the models and test the models using 660 observations.<br><br>The authors classify modules as risky (5 or more faults) and not risky (0-4 faults). | The authors observed a 23.8% type I error, a 13.8% type II error, and a 22.6% overall error.<br><br>The authors also construct a model without development information by not including the variables that indicate if a module is new or old and changed or unmodified. The authors observed a 32.4% type I error, a 21.3% type II error, and a 31.1% overall error. |

| | | | |
|---|---|---|---|
| Khoshgoftaar, Allen, Hudepohl, and Aud establish a relationship between content metrics and field problems using the discriminant analysis technique with principal component analysis and model selection and the neural networks techniques in 1997 [42].<br><br>The authors classify modules as risky or not risky. The study uses data from on 6972 modules in a release of a telecommunications software system. | The authors use predictors grouped in two classes:<br>Call graph metrics:<br>modules used, total calls to other modules, unique calls to other modules<br><br>Control flow graph metrics:<br>if-then conditional arcs, loops, nesting level, span of conditional arcs, span of loops, and McCabe Cyclomatic complexity.<br><br>The measure of field problems is faults. | The authors consider modules that are reused or changed.<br><br>The training set has 4638 modules and the testing set has 2324 modules. The authors classify modules as risky (3 or more faults) or not risky (0-2 faults).<br><br>The authors construct four principal components and use model selection to select three principal components for the discriminant analysis model. | The authors observed a 27.9% type I error, a 39.4% type II error, and a 29.5% overall error using the discriminant analysis model.<br><br>The authors use all four principal components for the neural networks model.<br><br>The authors observed a 26.0% type I error, a 26.9% type II error, and a 26.2% over all error using the neural networks model. |
| Takahashi, Muraoka, and Nakamura establish a relationship between content metrics and field problems using the trees technique in 1997 [96].<br><br>The authors classify modules as risky or not risky. The study uses data on 562 modules from a network management system. | The authors use predictors grouped into two classes:<br>Code metrics:<br>number of executable lines including data declarations on modified parts, number of comment lines on modified parts, Cyclomatic complexity on modified parts, number of executable lines including data declaration in entire module, number of commented lines in entire module, Cyclomatic complexity in entire module, the maximum nesting of C-code block in entire module, number of distinct operators in entire module, number of distinct operands in entire module, number of operator occurrences in entire module, number of operand occurrences in entire module, Halstead's Volume in entire module, Halstead's Difficult in entire module, Halstead's mental effort in entire module, reuse ratio | The authors normalize the metrics.<br><br>The authors use 393 observations to train the model and 169 observations to test the data.<br><br>The authors classify modules as risky (more than 20 faults per 1000 LOC) or not risky (less or equal to 20 faults per 1000 LOC). | The authors observed a 22% type I error, a 57% type II error, and a 28% overall error for the best tree. |

| | | | |
|---|---|---|---|
| | Structure metrics:<br>number of distinct functions in module that call this function, number of distinct functions in module called by this function, fan-in in module considering the number of called functions, fan-out in module considering the number of calling functions, average nesting of calling functions with decision node, number of predicates in modules including predicting defined on the subsystem whose scope extends to function in the subsystem, number of predicates in modules including predicting defined on the subsystem whose scope extends to function on other subsystems, number of predicates in modules excluding predicting defined on the subsystem whose scope extends to function in the subsystem, number of predicates in modules excluding predicting defined on the subsystem whose scope extends to function in other subsystems.<br><br>The measure of field problems is fault density. | | |
| Pighin and Zamolo establish a relationship between content metrics and field problems using a discriminant analysis technique in 1997 [89].<br><br>The authors classify modules as risky or not risky. The study uses data on 904 files in a management software system produced by a software house over a period of 7 years. | The authors use the predictors: preprocess instruction lines, # of define, # include, words of comment, new types, jump instructions, function arguments, assignment in function arguments, operators in function arguments, pointers in function arguments, function calls in function arguments, ternary if, casting operators, structure references, address operators, arithmetic operators, relational operators, logical operators, assignment operators, increment/decrement operators, operators, logical and relational in return expressions, external declarations, declarations of variable, declarations of function, | The authors construct principal components.<br><br>The authors divide the data into three groups, not risky (0 or 1 fault) risky (10 or more faults) and other (2-9 faults).<br><br>The training set contains 604 files and the testing set contains 300 files.<br><br>The ‘other’ group is not used in training (313 observations). | First the authors classify only modules that are risky or not risky in the testing set (discarding 148 observations).<br><br>The authors observed a 3.6% type I error, a 9.8% type II error, and a 5.3% overall error classifying observations using the model with all predictors.<br><br>The authors observed a 2.7% type I error, a 9.8% type II error, and a 4.6% overall error classifying observations using the model with 7 principal components. |

| | | | |
|---|---|---|---|
| | operations in array indexes, control instructions, iteration instructions, label instructions, max nesting with else, max nesting with deepness in control structure, function implemented in control structure, nesting level in control structure, relations in control structures, instructions in control structures, function calls, in control structure, lines in functions, time of observation.<br><br>The measure of field problems is faults. | | The authors observed a 4.5% type I error, a 0% type II error, and a 3.3% overall error classifying observations using the model with 10 principal components.<br><br>The authors redefine risky (more than 5 faults) and not risky (0-5 faults), and use a .80 classifying threshold to classify observations.<br><br>The authors observe a 7.8% type I error, a 26.1% type II error, and a 12.4% overall error for the model with all predictors, which allows 274 out of 300 observations (91.3%) to be classified.<br><br>The authors observe a 7.7% type I error, a 25.0% type II error, and an 11.5% overall error for the model with 7 principal components, which allows 252 out of 300 observations (84.0%) to be classified.<br><br>The authors observe a 7.0% type I error, a 16.1% type II error, and a 9.1% overall error for the model with 10 principal components, which allows 263 out of 300 observations (87.6%) to be classified. |
| Khoshgoftaar, Allen, Jones, and Hudepohl establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the linear modeling technique (logistic regression) with variable selection in 1998 | The authors use predictors grouped into two classes:<br><br>Software product metrics:<br>the number of distinct include files, log of the number of independent paths, maximum span of variables, proportion of systems with module installed,<br><br>Development process metrics:<br>number of problems in beta | The authors only examine modules that have been updated since the last release. | They observe a balance between type I error and type II error at a 27.71% type I error and a 22.96% type II error. |

| | | | |
|---|---|---|---|
| [49].<br><br>The authors classify modules as risky or not risky. The study uses data on “a few thousand modules” in a telecommunications system. | testing in previous release, total number of problems fixed in current release that originated from beta testing of previous release, total number of problems fixed in the current release found by customers in a prior release, net increase in lines of code due to software changes, number of different designers creating changes for this module, number of updates by designers who had 10 or less updates in company career, total number of updates that designers had in their company careers when they updated this module.<br><br>The measure of field problems is customer discovered faults. | | |
| Schenker and Khoshgoftaar establish a relationship between content metrics and field problems using the case-based reasoning technique in 1998 [91].<br><br>The authors classify modules as risky or not risky. The study uses data from 282 modules in a military telecommunications software system. | The predictors are: number of unique operators, number of unique operands, number of total operators, number of total operands, lines of code, executable lines of code, Cyclomatic complexity, and extended Cyclomatic complexity.<br><br>The measure of field problems is faults. | The training set has 188 modules and the testing set has 94 modules. The authors predict modules as risky (4 or more faults) or not risky (0-3 faults). | The authors observed a 10.7% type I error, a 31.6% type II error, and a 14.89% overall error. |
| Ebert establishes a relationship between content metrics and field problems using the heuristic technique, the trees technique, the discriminant analysis technique, the rule-based technique, and the neural networks technique in | The predictors are: executable statements, statement complexity (count of the uses of distinct statements), expression complexity (count of the uses of distinct expressions), data complexity (count of the uses of distinct statements for data types and their use), depth of nesting, database access.<br><br>The measure of field problems is | The training set has 251 modules and the testing set has 200 modules.<br><br>The author classifies modules are risky (more than 1 fault) or not risky (0 or 1 fault). | The author observes a 10.43% type I error, 45.95% type II error, and 18.5% overall error for the heuristic model.<br><br>The author observes an 8.59% type I error, 43.24% type II error, and 15% overall error for the trees model.<br><br>The author observes a 15.95% type I error, 32.43% type II |

| | | | |
|---|---|---|---|
| 1995/1997/1998 [13][14][15].<br><br>The author classifies modules as risky or not risky. The study uses data on 451 software components from several real-time telecommunications software system. | faults from testing and operation. | | error, and a 19% overall error for the discriminant analysis model.<br><br>The author observes an 18.40% type I error, 21.62% type II error, and a 19% overall error for the rule-based technique.<br><br>The author observes an 8.59% type I error, 43.24% type II error, and 15% overall error for the neural networks model. |
| Khoshgoftaar, Allen, Naik, Jones, and Hudepohl establish a relationship between content metrics, development metrics, and field problems using the trees technique in 1998 [50].<br><br>The authors classify modules as risky or not risky. The study uses data on "a few thousand modules" from a telecommunications software system. | The authors use predictors grouped in two main classes and three sub-classes under product metrics:<br>Product metrics- call graph metrics:<br>distinct calls to other modules, second and following calls to other modules,<br>Product metrics- control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br>Product metrics- statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used<br><br>Process metrics: | The authors construct "approximately equal" training and testing sets.<br>The authors classify modules that have been modified since the lat release as risky (have at least one defect) or not risky (no defects). | The authors observed a 26.2% type I error, and a 28.9% type II error for the model using both product and development measures.<br><br>The authors also constructed a tree using only product measures. The authors observed a 27.0% type I error and a 27.4% type II error. |

| | | | |
|---|---|---|---|
| | number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers, and proportion of systems with module installed.<br><br>The measure of field problems is faults. | | |
| Jones, Hudepohl, Khoshgoftaar, and Allen establish a relationship between content metrics, deployment and usage metrics, and field problems using the linear modeling (logistic regression) technique with variable selection in 1999 [32].<br><br>The authors classify modules as risky or not risky. The study uses data on "a few | The authors use predictors grouped into three classes:<br>Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of | The authors use model selection techniques to select the predictors: distinct include files, number of independent paths, proportion of systems with a module installed, and maximum span of variables.<br><br>The authors consider only modules that have been updated since the last release.<br><br>The authors classify modules as risky (at least one fault) or not risky (no faults)<br><br>The authors use "approximately | The authors observed a 29.06% type I error and a 30.77% type II error using the model with product and deployment and usage metrics.<br><br>The authors also construct a model without deployment and usage measures, by not including the proportion of systems with module installed variable. The authors observed a 27.32% type I error and a 34.24% type II error. |

| | | | |
|---|---|---|---|
| thousand modules" from a telecommunications system. | structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, and distinct variables used.<br><br>The measure of field problems is faults. | equal" training and testing sets. | |
| Khoshgoftaar and Allen establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique in 1999 [37].<br><br>The authors classify modules as risky or not risky. The study uses data on "a few thousand modules" from four releases of a telecommunications software system. | The authors use predictors grouped into five classes:<br>Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, | The authors consider only updated modules.<br><br>The use data from the first release as the training set and data on releases 2-4 as the testing set.<br><br>The authors classify modules as risky (1 or more faults) or not risky (0 faults). | The authors observed a 27.9% type I error and a 28.6% type II error for release 2.<br>The authors observed a 30.4% type I error and a 34.0% type II error for release 3.<br>The authors observed a 33.7% type I error and a 27.2% type II error for release 4. |

| | | | |
|---|---|---|---|
| | number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults during operation. | | |
| Khoshgoftaar, Allen, Yuan, Jones, and Hudepohl establish a | The author use predictors grouped into three main classes with three sub-classes under | The authors only consider modules that have been modified. The authors divide the data into a training set and a testing set but do | The authors observed a 30.30% type I error and a 23.81% type II error for model with product, development, and deployment |

|  |  |  |  |
|---|---|---|---|
| relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique in 1999 [51].<br><br>The authors classify modules as risky or not risky. The study uses data from two releases of a telecommunications software system (the authors do not reveal how many modules are examined). | software product metrics:<br>Software product metrics – Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Software product metrics - Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Software product metrics – Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code | not describe the process.<br><br>The authors classify modules as risky (1 or more faults) or not risky (0 faults).<br><br>The authors select the best model using a combined criterion of robustness (difference between fitted and test error rates), balance of accuracy (difference between type I and type II error), and parsimony (number of predictors used). | and usage metrics.<br><br>The authors also construct a tree with only product and development metrics. The authors observed a 23.55% type I error and a 32.80% type II error for the model with product, development, and deployment and usage metrics. |

| | | | |
|---|---|---|---|
| | for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults. | | |
| Khoshgoftaar, Allen, Jones, and Hudepohl establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique in 1999/2000 [43]/[44].<br><br>The authors classify modules as risky or not risky. The study uses data from four releases of a telecommunications software system (the | The authors use predictors grouped into five classes:<br>Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending | The authors only consider modules that have been modified or are new.<br><br>The tree is then used to classify modules as risky (1 or more faults) or not risky (0 faults) in the next three releases.<br><br>The authors have access to four releases. The authors construct two models. In the first model, the data from first release is used as the train set and the data from releases 2-4 are used as testing sets. In the second model, the second release is used to construct the tree.<br><br>The metrics used in the tree constructed using the first release | The authors observe a 27.9% type I error and a 28.6% type II error for release 2.<br>The authors observed a 30.4% type I error and a 34.0% type II error for release 3.<br>The authors observed a 33.7% type I error and a 27.2% type II error for release 4.<br><br>The authors observe a 27.4% type I error and a 27.6% type II error for release 3.<br>The authors observed a 31.5% type I error and a 35.9% type II error for release 4. |

|  |  |  |  |
|---|---|---|---|
| authors do not reveal how many modules are examined). | nodes, log of number of independent paths<br><br>Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release,  number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module,  net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics:<br>proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average | are: number of distinct include files, proportion of systems with modules deployed, number of different designers making changes, new and changed lines of code, maximum span of variables, number of update that designers had in their company careers, number of entry nodes, number of internal nodes, total number of changes to the code, number of control statements, and span of branches of conditional arcs.<br><br>The metrics used in the tree constructed using the second release are: maximum span of variables, number of distinct include files, number of changes to the code for any reason, maximum control structure nesting, number of distinct variables used, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, maximum span of branches of conditional arcs, number of problems fixed that were found by customers in the prior release, number of second and following uses of variables, proportion of systems with modules deployed, total span of variables, net new and changed lines of code, log of the number of independent paths. |  |

| | | | |
|---|---|---|---|
| | transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults during operation. | | |
| Khoshgoftaar, Allen, and Busboom establish a relationship between content metrics and field problems using the case-based reasoning technique, the clustering technique, and the linear modeling technique with heuristics (linear regression) in 2000 [39].<br><br>The authors classify modules as risky or not risky. The study uses data from 282 modules in a telecommunications software system. | The author use predictors grouped into three main classes with three sub-classes under software product metrics:<br>Software product metrics – Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Software product metrics - Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Software product metrics – Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems | The authors divide the data into a training set (188 modules) and a testing set (94 modules).<br><br>The authors classify modules as risky (4 or more faults) or not risky (0-3 faults). | The authors observed a 16.0% type I error and a 15.8% type II error using case based reasoning model. The authors observed a 14.7% type I error and a 21.1% type II error using data clustering model. The authors observed a 16.0% type I error and a 15.8% type II error using the linear modeling model (selecting the top 30% of modules as risky). |

| | | | |
|---|---|---|---|
| | found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults. | | |
| Khoshgoftaar, Thanker, and Allen establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique with | The authors use predictors grouped into five classes:<br>Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br><br>Control flow graph metrics:<br>arcs that are not conditional, non | The authors consider only updated and new modules.<br><br>The authors have access to four releases. The authors use data from release 1 as the training set and data from releases 2-4 as the testing set. | The best tree for the entire system according the authors is the tree built using product and development principal component analysis and execution metrics. It produces a 29.3% type I error and 21.2% type II error for release 2, 29.9% type I error and 19.1% type II error for release 3, and 32.7% type I error and 19.6% type II |

| | | | |
|---|---|---|---|
| principal component analysis in 2000 [64].<br><br>The authors classify modules as risky or not risky. The study uses data from four releases of a telecommunications software system (the authors do not reveal how many modules are examined). | loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had | The authors predict modules as risky (1 or more faults) or not risky (0 faults). The authors consider training the tree using only modules in a subsystem to classify all modules and training the tree using all modules to classify modules in a subsystem.<br><br>The authors also use different combinations of categories of metrics with and with out principal component analysis: product and execution metrics using all modules, product and execution metrics using only modules in a subsystem, product, process, and execution metrics using all modules, product, process, and execution metrics using only modules in a subsystem, product metrics with principal component analysis and execution metrics using all modules, product metrics with principal component analysis and execution metrics using only modules in a subsystem, product and execution metrics with principal component analysis and execution metrics using all modules, product and execution metrics with principal component analysis and execution metrics using only modules in a subsystem. | error for release 4.<br>The best tree for the sub systems according to the authors is the tree built using product principal components and deployment and usage metrics. It produces a 30.1% type I error and 28.8% type II error for release 2, 34.2% type I error and 13.6% type II error for release 3, 36.9% type I error and 31.1% type II error for release 4.<br><br>The authors conclude that the model built using principal component analysis perform better than trees built using raw metrics.<br>The authors conclude that a model built using only data from the subsystem is more accurate than a model built using data from the entire system when predicting for the subsystem all else being equal.<br>The authors conclude that the model built using data from the entire system is better than the model built using data from a subsystem when predicting for the entire systems all else being equal. |

| | | | |
|---|---|---|---|
| | between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults during operation. | | |
| Khoshgoftaar, Shan, and Allen establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique with principal component analysis in 2000 [62]/[63].<br><br>The authors classify modules as risky or not risky. The study uses data on “a few thousand modules” from four releases of a telecommunications software system. | The author use predictors grouped into three main classes with three sub-classes under software product metrics:<br>Software product metrics – Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br>Software product metrics - Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br>Software product metrics – Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of | The authors only consider new or updated modules.<br><br>The authors have access to four releases. The authors use data from release 1 as the training set and data from releases 2-4 as the testing set.<br><br>The authors predict modules as risky (1 or more faults) or not risky (0 faults).<br><br>The authors use principal component analysis to construct 6 product related principal components and 8 development related principal components (keeping the four deployment and usage metrics unchanged). | The authors observe that the best model full model has 25.32% type I error and 23.81% type II error for release 2, 27.36% type I error and 19.15% type II error for release 3, 25.71% type I error and 27.17% type II error for release 4.<br><br>The authors also constructed a tree with out the development principal components and observe 24.76% type I error and 25.93% type II error for release 2, 28.25% type I error and 29.79% type II error for release 3, 35.59% type I error and 21.74% type II error for release 4. |

| | | | |
|---|---|---|---|
| | variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release,  number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module,  net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics:<br>proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults. | | |
| Khoshgoftaar, Allen, and Deng | The authors use predictors grouped into two main classes | The authors only consider new or | The best predictive model (not the best fitted model) produces a |

| | | | |
|---|---|---|---|
| establish a relationship between content metrics, deployment and usage metrics, and field problems using the trees technique in 2001 [41].<br><br>The authors classify modules as risky or not risky. The study uses data on “a few thousand modules” from four releases of a telecommunications software system. | with two sub-classes under software product metrics:<br>Software product metrics – Call graph metrics:<br>total number of calls, distinct calls to other modules, second and following calls to other modules<br><br>Software product metrics – Control flow graph metrics:<br>arcs that are not conditional, total span of branches of conditional arcs, maximum span of branches of conditional arcs, maximum control structure nesting, number of distinct include files, non loop conditional arcs, knots (where arcs cross due to a violation of structured programming principal), log of number of independent paths, lines of code, loop constructs, entry nodes, exit nodes, internal nodes, pending nodes, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, total number of variables, distinct variables used, number of second and following uses of variables<br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults discovered by customers. | updated modules.<br><br>The authors have access to four releases. The authors use data from release 1 as the training set and data from releases 2-4 as the testing set.<br><br>The authors predict modules as risky (1 or more faults) or not risky (0 faults). | 27.93% type I error and a 25.93% type II error for release 2.<br>The authors observed a 26.79% type I error and a 17.02% type II error for release 3.<br>The authors observed a 33.17% type I error and a 20.65% type II error for release 4. |
| Kokol, Podgorelec, Zorman, Sprogar, and Pighin establish | The authors report calculating 168 different metrics but do not give the list of predictors. | The authors use a training set and a testing set (the authors do not reveal how many modules are in | The authors observe the best discriminant analysis model (produced using 10 principal |

| | | | |
|---|---|---|---|
| a relationship between content metrics, development metrics, and field problems using the discriminant analysis technique with principal component analysis, the linear modeling technique, the linear programming technique, the trees technique, and the neural networks with trees technique in 2001 [66].<br><br>The authors predict classify modules as risky or not risky. The study uses data on 911 modules from a hospital information system. | The measure of field problems is faults. | each set). | components) classifies 87.6% of the modules and has a 7.0% type I error, a 16.1% type II error, and a 9.1% overall error.<br>The authors observe the linear model using Cyclomatic complexity classifies 100% of modules and has a 19.7% type I error, a 34.3% type II error, and a 24.2% overall error.<br>The authors observe the linear model using the alpha metric classifies 100% of modules and has an 18.1% type I error, a 35.8% type II error, and a 30.9% overall error.<br>The authors observe the linear model using the RPSM metric classifies 100% of modules and has a 31.5% type I error, a 14.9% type II error, and an 18.6% overall error.<br>The authors observe the linear programming model using the RPSM metric classifies 100% of modules and has a 32.6% type I error, an 8.4% type II error, and a 16.0% overall error.<br>The authors observe the one trees model classifies 100% of modules and has a 13.7% type I error, 55.6% type I error, and a 25% overall error.<br>The author observe a second trees model classifies 100% of modules and has a 15.1% type I error, a 22.2% type II error, and a 17% overall error.<br>The authors observe a neural networks and trees model classifies 100% of modules, has a 28.5% type I error, a 14.8% type II error, and a 25% overall error. |
| Pighin, Podgorelec, and Kokol establish a relationship between content metrics and field problems using the linear programming | The authors report collecting 220 predictors, use statistical reduction to reduce the number of metrics to 149, finally, using parameter selection, they use 29 metrics. | The authors divide the data into a learning set (200 modules) and a testing set (172 modules).<br><br>The authors classify modules as risky (6 or more faults) and not | The authors observe a 32.59% type I error and 8.42% type II error. |

| | | | |
|---|---|---|---|
| technique in 2002 [90].<br><br>The authors classify modules as risky or not risky. The study uses data on 372 files from a management software system produced by a software house over a period of 7 years. | The reported predictors include: preprocess instruction lines, # of define, # include, words of comment, new types, jump instructions, function arguments, assignment in function arguments, operators in function arguments, pointers in function arguments, function calls in function arguments, ternary if, casting operators, structure references, address operators, arithmetic operators, relational operators, logical operators, assignment operators, increment/decrement operators, operators, logical and relational in return expressions, external declarations, declarations of variable, declarations of function, operations in array indexes, control instructions, iteration instructions, label instructions, max nesting with else, max nesting with deepness in control structure, function implemented in control structure, nesting level in control structure, relations in control structures, instructions in control structures, function calls, in control structure, lines in functions, time of observation.<br><br>The measure of field problems is faults. | risky (0-5 faults). | |
| Ohlsson and Runeson establish a relationship between content metrics and field problems using the discriminant analysis technique with principal component analysis in 2002 [85].<br><br>The authors classify modules as risky or not risky. The study uses data on 28 | The predictors are: number of output symbols in design doc, number of input symbols in design doc, number of unique external outputs from a components, number of unique external inputs to a component, number of state symbols in design doc, Cyclomatic complexity adapted for design doc, number of design pages in design doc, number of task symbols in the design doc, number of if boxes in the design doc. | The authors have access to two releases. The authors use data from release 1 as the training set and data from releases 2 as the testing set.<br><br>The authors predict modules as risky (10 or more faults) or not risky (less than 10 faults).<br><br>The authors use principal component analysis to construct 2 principal components. | The authors observe that the best model has 18%% type I error, 27%% type II error, and 21% overall error. |

| | | | |
|---|---|---|---|
| software components from two releases of a real-time telecommunications software system. | The measure of field problems is fault reports from test and operation. | | |
| Khoshgoftaar, Allen, and Deng establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique in 2002 [40].<br><br>The authors classify modules as risky or not risky. The study uses data on “a few thousand modules” from four releases of a telecommunications software system. | The authors use predictors grouped into three classes:<br>Call graph metrics:<br>number of distinct calls to other modules, number of second and following calls to other modules<br><br>Software product metrics – Control flow graph metrics:<br>number of arcs that are not conditional, number of non-loop conditional arcs, number of loop constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), number of internal nodes, number of entry nodes, number of exit nodes, number of pending nodes, log of number of independent paths<br><br>Software product metrics – Statement metrics:<br>number of distinct include files, number of lines of code, number of control statements, number of declarative statements, number of executable statements, number of global variables used, total span of variables, maximum span of variables, number of distinct variables used, number of second and following uses of variables<br><br>The measure of field problems is faults discovered by customers. | The authors only consider new or updated modules. The authors have access to four releases. The authors use data from release 1 as the training set and data from releases 2-4 as the testing set.<br><br>The authors predict modules as risky (1 or more faults) or not risky (0 faults).<br><br>The authors find the most balanced tree (minimum difference between type I and type II errors). | The authors observed a 25.1% type I error and a 26.5% type II error for release 2.<br>The authors observed a 26.7% type I error and a 21.3% type II error for release 3.<br>The authors observed a 32.2% type I error and a 21.7% type II error for release 4. |
| Khoshgoftaar and Seliya establish a relationship between | The authors use predictors grouped into four classes:<br>Product metrics – Call graph | The authors classify modules as risky (1 or more faults) or not risky (no faults). The authors use | The authors observed for the SPRINT tree a 24.47% type I error and a 26.84% type II error |

| | | | |
|---|---|---|---|
| content metrics, development metrics, and field problems using the trees techniques in 2002 [59].<br><br>The authors classify modules as risky or not risky. The study uses data on 3649 modules, 3981 modules, 3541 modules, and 3978 modules from four consecutive releases of a telecommunications software system. | measures:<br>distinct calls to other modules, second and following calls to other modules<br><br>Product metrics – control flow graph measures:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Product metrics – statement measures:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables,<br><br>Execution metrics:<br>proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults discovered by customers resulting in code change. | release 1 data to train the models and data on releases 2-4 to test the models.<br><br>The authors compare models built using two algorithms (SPRINT and CART) using multiple criterions.<br><br>The SPRINT model as 15 nodes, while the CART model has 3 nodes.<br><br>The authors also evaluate the complexity, (the number of nodes), balance (difference between type I and type II errors), and stability (changes in the balance) of the models to decide that the SPRINT trees are better. | for release 2.<br>The authors observed a 25.38% type I error and a 20.83% type II error for release 3.<br>The authors observed a 28.48% type I error and a 26.88% type II error for release 4.<br><br>The authors observed for the CART tree a 31.67% type I error and a 23.28% type II error for release 2.<br>The authors observed a 30.30% type I error and a 14.89% type II error for release 3.<br>The authors observed a 35.64% type I error and a 22.82% type II error for release 4. |
| Ostrand, Weyuker, | The predictors are: The authors | The authors use a negative | The authors observe 20%, 29%, |

| | | | |
|---|---|---|---|
| and Bell establish a relationship between content metrics, development metrics, and field problems using a linear modeling technique (negative binomial technique) with variable selection and a heuristic technique in 2003 [87].<br><br>The authors classify files as risky or not risky. The study uses data on 1950 files from 17 releases of an inventory system and 2271 files from 9 releases of a provisioning system at AT&T. | use a model selection technique to select the predictors: lines of code, a files change status, a file's age, the number of faults identified during the previous release, the programming language used, and the release number, number of changes since previous release, file changed prior to previous release, log of the Cyclomatic complexity.<br><br>The measure of field problems is defect MRs (modification request). | binomial model to predict the number of defects in a file, and then use a Pareto distribution (top 20%) to identify the risky files.<br><br>For the inventory system, the authors use data from the release 1-2 as the training set and data from subsequent releases as the testing set.<br><br>For the provisioning system, the authors create 3 releases, out of release 1 (release A), releases 2-5 (release B), and releases 6-9 (release C). The authors use data from release A and B as training sets and data from release C as the testing set. | 27%, 10%, 19%, 17%, 19%, 19%, 17%, 10%, 10%, 7%, 15%, 12%, and 14% type II error for releases 3-17 (an average of 16% type II error).<br><br>The authors also constructed a simplified model using only the lines of code predictor. The authors observe 29%, 32%, 29%, 25%, 24%, 24%, 23%, 23%, 20%, 25%, 32%, 20%, 29%, 33%, 42% type II errors for releases 3-17 (an average of 27% type II error).<br><br>The authors observe a 17% type II error for the provisioning system. |
| Khoshgoftaar, Geleyn, and Nguyen establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the trees technique with bagging, boosting, and logitboost techniques in 2003 [53].<br><br>The authors classify modules as risky or not risky. The study uses data on 3649, 3981, 3541, and 3978 modules from four release of a telecommunications software system.<br><br>The authors also study two wireless | The authors use predictors grouped into five classes:<br><br>Call graph metrics:<br><br>distinct calls to other modules, second and following calls to other modules<br><br>Control flow graph metrics:<br><br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br><br>Statement metrics:<br><br>distinct include files, lines of code, control statements, declarative statements, | The authors build kinds of trees: a full tree model and a decision stomp model (tree with only 2 levels). | The authors observed a 26.81% type I error, a 30.69% type II error, and a 27.0% overall error using the full tree for release 2.<br><br>The authors observed a 24.83% type I error, a 28.57% type II error, and a 25.01% overall error using the full tree with bagging for release 2.<br><br>The authors observed a 31.24% type I error, a 30.69% type II error, and a 31.21% overall error using the full tree with boosting for release 2.<br><br>The authors observed a 30.91% type I error, a 23.40% type II error, and a 30.81% overall error using the full tree for release 3.<br><br>The authors observed a 29.91% type I error, a 31.9% type II error, and a 29.94% overall error using the full tree with bagging for release 3.<br><br>The authors observed a 36.15% type I error, a 21.28% type II error, and a 35.95% overall error using the full tree with |

| | | | |
|---|---|---|---|
| configurations products. However, the two systems do not have a measure of field problems (they use faults during system test). We do not report results on the two wireless products.<br><br>The predictors, model building procedure, and the classification criterions are the same as in [43]. | executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables<br><br>Software process metrics:<br>number of problems found by designers, number of problems found during beta testing, number of problems fixed that were found by designers, number of problems fixed that were found by beta testing in the prior release, number of problems fixed that were found by customers in the prior release, number of changes to the code due to new requirements, total number of changes to the code for any reason, number of distinct requirements that caused changes to the module, net increase in lines of code, net new and changed lines of code, number of different designers making changes, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates to this modules by designers how had between 11 and 20 total updates in entire company career, number of updates that designers had in their company careers<br><br>Software execution metrics: proportion of systems with a module installed, execution time of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults during operation. | | boosting for release 3.<br>The authors observed a 32.84% type I error, a 21.74% type II error, and a 32.58% overall error using the full tree for release 4.<br>The authors observed a 34.71% type I error, a 16.30% type II error, and a 34.29% overall error using the full tree with bagging for release 4.<br>The authors observed a 36.46% type I error, a 26.09% type II error, and a 36.22% overall error using the full tree with boosting for release 4.<br><br>The authors observed a 35.25% type I error, a 20.63% type II error, and a 34.55% overall error using the decision stomp for release 2.<br>The authors observed a 35.25% type I error, a 20.63% type II error, and a 34.55% overall error using the decision stomp with bagging for release 2.<br>The authors observed a 28.35% type I error, a 24.34% type II error, and a 28.16% overall error using the decision stomp with boosting for release 2.<br>The authors observed a 22.61% type I error, a 29.63% type II error, and a 22.95% overall error using the decision stomp with logitboost for release 2.<br>The authors observed a 33.26% type I error, a 14.89% type II, and a 33.01% overall error using the decision stomp for release 3.<br>The authors observed a 33.26% type I error, a 14.89% type II error, and a 33.01% overall error using the decision stomp with bagging for release 3.<br>The authors observed a 31.31% type I error, a 14.89% type II error, and a 31.09% overall |

| | | | |
|---|---|---|---|
| | | | error using the decision stomp with boosting for release 3.<br><br>The authors observed a 26.45% type I error, a 14.89% type II error, and a 26.09% overall error using the decision stomp with logitboost for release 3.<br><br>The authors observed a 36.36% type I error, a 22.83% type II error, and a 36.05% overall error using the decision stomp for release 4.<br><br>The authors observed a 36.36% type I error, a 22.83% type II error, and a 36.05% overall error using the decision stomp with bagging for release 4.<br><br>The authors observed a 36.62% type I error, an 18.48% type II error, and a 36.20% overall error using the decision stomp with boosting for release 4.<br><br>The authors observed a 26.38% type I error, a 23.91% type II error, and a 26.32% overall error using the decision stomp with logitboost for release 4.<br><br>The authors conclude that the best model is constructed using decision stomps with logitboost. |
| Khoshgoftaar, and Seliya establish a relationship between content metrics and field problems using the sets technique in 2004 [61].<br><br>The authors classify modules as risky or not risky. The study uses data on 282 modules from a military telecommunications software system. | The predictors are: number of unique operators, number of unique operands, number of total operators, number of total operands, lines of code, executable lines of code, Cyclomatic complexity, and number of logical operators.<br><br>The measure of field problems is faults during systems test and first year of operation. | The authors use data on 188 modules to as the training set and data on 94 modules as the testing set. The authors classify modules as risky (4 or more faults) or not risky (1-3 faults). | The authors observed a 14.67% type I error and a 10.53% type II error. |

## 5.3 What is the number of field problems?

This section reviews research work that addresses the question what is the number of field problems. The studies are summarized in Table 13.

**Table 13. Summary of research work that address what is the number of field problems**

| Research overview | Predictors and field problem metric | Modeling specific details | Results |
|---|---|---|---|
| Khoshgoftaar, Bhattacharyya, and Richardson establish a relationship between content metrics and field problems using the non-linear regression technique and the linear modeling technique with model selection in 1992 [52].<br><br>The authors predict what the number of field problems is. The authors use data from 138 modules from a military communications system and data on 20 modules from a previous study. | The predictor for the previous study is lines of code.<br><br>The predictors for the military system are: number of unique operators, number of unique operands, number of total operators, number of total operands, program length, program volume, Halstead's effort metric, Cyclomatic complexity, extended Cyclomatic complexity, number of procedure calls, number of comment lines, number of blank lines, number of lines of code, and number of executable lines of code.<br><br>The measure of field problems is faults. | The authors fit a non-linear model to the data from a previous study.<br><br>The authors use 15 modules as the training set and 5 observations as the testing set.<br><br>The authors fit a linear model with model selection for the military data.<br><br>The authors use 92 modules as the training set and 46 modules as the testing set. | The authors observe an average relative error of .505 for a model fitted using minimum relative error. The average relative error is .537 for a model fitted using relative least squares. The average relative error is .754 for a model fitted using least absolute value. The average relative error is .741 for a model fitted using least squares.<br><br>The authors observe an average relative error of .585 for a model fitted using minimum relative error. The average relative error is .594 for a model fitted using relative least squares. The average relative error is .747 for a model fitted using least absolute value. The average relative error is .757 for a model fitted using least squares. |
| Khoshgoftaar, Pandya, and More establish a relationship between content metrics and field problems using the linear modeling technique (linear regression) with model selection and the neural networks technique in 1992 [58].<br><br>The authors predict what number of field problems is. The authors use data from 282 modules | The predictors are: number of unique operators, number of unique operands, number of total operators, number of total operands, program length, program volume, Halstead's effort metric, Cyclomatic complexity, extended Cyclomatic complexity, number of procedure calls, number of comment lines, number of blank lines, number of lines of code, and number of executable lines of code.<br><br>The measure of field problems is faults during systems testing and the first year of operation. | The authors use 188 modules as the training set and the remaining 94 modules as the testing set. | The authors observe an average relative error of .5877 with a standard deviation of .6248 for the linear regression model with model selection.<br><br>The authors observe an average relative error of .3980 with a standard deviation of .2786 for the neural networks model. |

| | | | |
|---|---|---|---|
| from a military communications system. | | | |
| Khoshgoftaar, Munson, and Lanning establish a relationship between content metrics and field problems using the linear technique with the clustering and principal component analysis in 1993 [56].<br><br>The authors predict what the number of field problems is. The authors use data from 390 modules from a medical imaging system. | The predictors are: lines of code, lines of code excluding comments, number of characters, number of code characters, program length, Jenson's estimator of program length, Cyclomatic complexity and Belady' bandwidth.<br><br>The measure of field problems is the number of changes due to faults found during systems testing and maintenance. | The authors use 260 modules as the training set and 130 modules as the testing set.<br><br>The authors construct two principal components and cluster modules using the two components.<br><br>The authors construct four clusters, and decide to disregard one of the clusters that consist of outliers.<br><br>The authors fit separate models for each cluster as well as one model for all the modules. | For the first cluster consisting of 17 modules the cluster specific model has an average absolute error of 14.98 while the general model has an average absolute error of 14.82.<br>For the second cluster consisting of 45 modules the cluster specific model has an average absolute error of 2.79 while the general model has an average absolute error of 3.36.<br>For the third cluster consisting of 70 modules the cluster specific model has an average absolute error of 3.65 while the general model has an average absolute error of 4.69.<br>Over all the cluster specific models has an average absolute error of 4.58 while the general model has an average absolute error of 5.32. |
| Khoshgoftaar, Pandya, and Lanning establish a relationship between content metrics field problems using a linear modeling technique and a neural networks technique in 1995 [57].<br><br>The authors predict what the number of field problems is. The authors use data from 282 modules from a military communications system and 339 modules from a medical imaging system. | The predictors for the military system are: number of unique operators, number of unique operands, number of total operators, number of total operands, lines of code, executable lines of code, McCabe's Cyclomatic.<br><br>The measure of field problems is faults during testing and fist year of development.<br><br>The predictors for the medical imaging system are: lines of code including comments, lines of code, total character count, tot comments, number of comment characters, number of code characters, Halstead's program length, Halstead's estimated program length, Jenson's estimator of program length, Cyclomatic complexity, and | For the military system, the authors use 188 modules as the training set and 94 modules as the testing set.<br><br>For the medical imaging system, the authors use 226 modules as the training set and 113 modules as the testing set. | For the military system, the authors observe a .5877 absolute relative error and a .62 standard deviation for the military system using the linear regression with model selection model.<br>The authors observe a .3980 absolute relative error and a .28 standard deviation for the military system using the neural networks model.<br><br>For the medical imaging system, the authors observe a .9998 absolute relative error and a 1.37 standard deviation for the medical imaging system using the linear regression with model selection model.<br>The authors observe a .5467 absolute relative error and a .08 standard deviation for the medical imaging system using the neural networks model. |

| | | | |
|---|---|---|---|
| | Belady's bandwidth metric.<br><br>The measure of field problems is the number of changes due to faults during testing and maintenance. | | |
| Khoshgoftaar, Allen, Kalaichelvan, and Goel establish a relationship between content metrics, development metrics, and field problems using the linear modeling technique (linear regression) principal component analysis and the discriminant analysis technique in 1996 [46].<br><br>The authors classify modules as risky or not risky and predict what the number of field problems is. The study uses data on 1980 modules from one release of a telecommunications system. | The predictors are: modules used, total calls to other modules, unique calls to other modules, if-then conditional arcs, loops, nesting level, span of conditional arcs, span of loops, McCabe Cyclomatic complexity, if a module is new, if a module is modified.<br><br>The measure of field problems is faults from coding through operations. | The authors use 1320 modules as the training set and 660 modules as the testing set.<br><br>The authors construct principal components from the product measures.<br><br>The authors use variable selection to select three principal components and both indicator variables.<br><br>The authors classify modules as risky (5 or more faults) and not risky (1-4 faults). | The authors observed a 23.8% type I error, a 13.8% type II error, and a 22.6% overall error for the discriminant analysis model.<br><br>The authors predict the number of faults using a linear regression model. The authors observe an average absolute error of 1.68 and an average relative error of 56.5%. |
| Yuan, Khoshgoftaar, Allen, and Ganesan establish a relationship between content metrics, development metrics, deployment and usage metrics, and field problems using the clustering technique and the linear modeling technique in 2000 [103].<br><br>The authors classify modules as risky or not risky and predict | The predictors are: number of distinct include files, log of the number of independent paths, maximum span of variables within a procedure, number of problems found during beta testing, number of problems fixed that were found by beta testing in the prior release, net increase in lines of code due to software changes, number of different designer who updated this module, number of updates to this module by designer how had 10 or less updates in entire company career, number of updates that designers had in their company careers, proportion of systems with | The authors examine only new or modified modules. The authors classify modules as risky (1 or more faults) or not risky (0 faults).<br><br>The authors create clusters then use model modules within the cluster using the linear modeling technique.<br><br>The authors rank the modules according to their predicted number of defects and select the top 70% as defect prone.<br><br>The authors build another module using the number of distinct | The authors observe an average absolute error of .1472, an average relative error of .1051, a 26.76% type I error, and a 29.63% type II error for the model using all predictors except maximum span of variables.<br><br>The authors observe an average absolute error of .1607, an absolute relative error of .1139, a 26.76% type I error and a 29.63% type II error for the simplified model. |

| | | | |
|---|---|---|---|
| what the number of field problems is. The study uses data from a telecommunications software system (the authors do not reveal how many modules are examined). | module installed.<br>The authors use these predictors because they are significant predictors in prior work.<br><br>The measure of field problems is faults discovered by the customer. | include files, the log of the number of independent paths, and the proportion of systems with modules deployed | |
| Xu, Khoshgoftaar, and Allen establish a relationship between content metrics, deployment and usage metrics, and field problems using the neural networks techniques with principal component analysis in 2000 [102].<br><br>The authors predict what the number of field problems is. The study uses data from four releases of a telecommunications software system (the authors do not reveal how many modules are examined). | The authors use predictors grouped into two main classes with three sub-classes under software product metrics:<br>Software product metrics – Call graph metrics:<br>distinct calls to other modules, second and following calls to other modules<br>Software product metrics – Control flow graph metrics:<br>arcs that are not conditional, non loop conditional arcs, loops constructs, total span of branches of conditional arcs, maximum span of branches of conditional arcs, total control structure nesting, maximum control structure nesting, knots (where arcs cross due to a violation of structured programming principal), internal nodes, entry nodes, exit nodes, pending nodes, log of number of independent paths<br>Software product metrics – Statement metrics:<br>distinct include files, lines of code, control statements, declarative statements, executable statements, global variables used, total span of variables, maximum span of variables, distinct variables used, number of second and following uses of variables,<br><br>Software execution metrics: proportion of systems with a module installed, execution time | The authors use data from the first release as the training set and data from the second release as the testing set.<br><br>The authors use principal component analysis to produce six principal components. | The authors observe a .74 absolute average error for release 2, a .85 absolute average error for release 3, a .90 absolute average error for release 4. |

| | | | |
|---|---|---|---|
| | of an average trans action on a system serving customers, execution time of an average transaction on a systems serving businesses, and execution time of an average transaction on a tandem system.<br><br>The measure of field problems is faults after unit testing through operations. | | |
| Graves, Karr, Marron, and Siy establish a relationship between content metrics and field problems using the linear technique with model selection and the non-linear regression technique in 2000 [22].<br><br>The authors predict what the number of field problems is. The study uses data on 80 modules from a release of a communication software system. | The predictors are lines of code, deltas, and module age. The authors also consider Cyclomatic complexity, functions, breaks, unique operators, total operands, program volume, expected length, variable count, max span, mean span, program level but show the authors do not consider the predictors because they are highly correlated with lines of code.<br><br>The measure of field problems is fault related modification requests. | The authors find that the best linear model is the one constructed using deltas and the age of the module.<br><br>The authors construct a sTable model, which assumes the number of changes remain constant between time intervals.<br><br>The authors also construct a non-linear model with lines of code. | The sTable model has an error of 757.4.<br><br>The best linear model produces an error of 697.4 , which the authors deem to be not significant.<br><br>The non-linear model produces an error of 631 which the authors conclude is significant. |
| Khoshgoftaar and Seliya establish a relationship between content metrics and field problems using the trees technique in 2002 [60].<br><br>The authors predict what the number of field problems is. The study uses data on 6972 modules from a telecommunications software system. | The authors use predictors grouped into two classes:<br>Call gram metrics:<br>unique procedure calls, total calls to others, distinct include files<br><br>Control flow graph metrics:<br>Cyclomatic complexity, number of loops, number of if-then constructs, total nesting level, total number of vertices within the span of loops or if-then statements, total edges plus vertices within loops structures.<br><br>The measure of field problems is faults leading to code change during testing and operations. | The authors only consider modules that are new or have been changed.<br><br>The authors use 4846 modules as the training set and 2324 modules as the testing set. The authors construct three different types of trees. | The authors observe a 1.1308 average absolute error and a .3943 average relative error for a CART-tree constructed using least absolute difference.<br><br>The authors observe a 1.2889 average absolute error and a .6845 average relative error for a S-Plus tree (construct using deviance).<br><br>The authors observe a 1.2894 average absolute error and a .6853 average relative error for a CART-tree constructed using least squares. |

| | | | |
|---|---|---|---|
| Mockus, Zhang, and Li establish a relationship between deployment and usage metrics, software and hardware configurations metrics, and field problems using the linear modeling technique in 2005 [74].<br><br>The study uses data from a telecommunications software system (the authors do not reveal how many systems are examined). The authors classify observations as risky or not risky and predict what the number of field problems is. | The predictors are: system size, operating system, number of ports, total deployment time, and software upgrades.<br><br>The measures of field problems are customer calls, technician dispatches, alarms, and software defects reported by customers. | The authors also predict the likelihood of a software failure using a logistic regression model.<br><br>The authors predict the number of customer calls, technician dispatches, and alarms using a linear regression model. | The authors show that the prediction of the number of customer calls is good graphically. |

## 6. 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.

## 7. 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.

## 8. 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] Kathyrn 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 Buckly 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*. Chapmann & 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 Edware 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 ISSTA,* 2002.

[87] Thomas J. Ostrand and Elaine J. Weyuker and Thomas Robert M. Bell. Where the bugs are. In *Proceedings of ISSTA,* 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 YoichiMuraoka 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.