A Catalog of Techniques that Predict Information about the Count or Rate of Field Defects

Paul Luo Li
December 2006
CMU-ISRI-06-122

Institute for Software Research, International
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

This work has been funded in part by the EUSES Consortium via the National Science Foundation (ITR-0325273), by the National Science Foundation under Grant CCF-0438929, by the Sloan Software Industry Center at Carnegie Mellon, and by the High Dependability Computing Program from NASA Ames cooperative agreement NCC-2-1298.

Keywords: Software engineering, management, measurement, metrics, software quality assurance, reliability, risk management, planning, software reliability growth models, reliability modeling, statistical models, defect prediction.

Abstract

Quality of software in the field is an important concern for producers of software, who often need to predict information about the count or rate of field defects to perform activities to manage the quality of their software products. To help software producers select appropriate techniques for making such predictions, we provide a catalog of techniques that are commonly used in the literature for predicting information about the count or rate of field defects. This catalog presents information on the intuition behind each technique and its inputs, outputs, procedures, applicability, cost of use, and quality of predictions. Finally, we discuss promising research that addresses some of the problems with the techniques that are commonly used today.

Table of contents

1. Introduction

Software producers often need to predict information about the count or rate of field defects to manage the quality of their software products [11]. The count of field defects and the rate of field defects are commonly used measures of the quality of software in the field, as discussed by Chulani et al. in [9]. Reliability, which is another common measure of quality, is the inverse of the count of field defects remaining in the software [20]. We use *field defect* to refer to all the terms used in the literature to describe a software related quality problem that occurs after release, such as a fault or a failure. Software producers commonly use the predicted information on the count or rate of field defects to:

- Decide whether to conduct more testing before release [63],
- Allocate resources for maintenance [32], and/or
- Guide process improvement efforts [1].

A particularly common kind of software today is multi-release software, for which software producers may need to make predictions for each release of the software.

Not all techniques in the literature that predict information about the count or rate of field defects are likely to be useful to software producers; therefore, we consider techniques with the following characteristics:

- The techniques use measures of the software, i.e. software metrics, as inputs.
    - The inputs used by techniques in the literature to make predictions are software metrics and/or expert opinion. Predictions made using expert opinion are less reliable than predictions made using software metrics because expert opinion is subjective, as discussed by Madridakis and Wheelwright in [57]. Furthermore, predictions made using software metrics are easier to analyze for decision making than predictions made using expert opinion, as discussed by Chulani in [8].
- The techniques can make predictions before the time of release.
    - Most activities that require the predicted information, such as deciding whether to conduct more testing before release and allocating resources for maintenance, need the information before release.
- The techniques have been used to make predictions for real-world software.
    - We only include techniques that have been used to make predictions for real-world software because techniques that have not been used to make predictions for real-world software may have unforeseen problems making predictions.
- The techniques that have users who are not of the group of people that developed the techniques.
    - We only include techniques that have users who are not of the group of people that developed the techniques because techniques that have only been used by the group of people that developed the techniques may have unforeseen problems making predictions.

The techniques that we consider fall into two general categories: software reliability growth model (SRGM)-based modeling techniques and statistical modeling techniques. SRGM-based modeling techniques are generally applicable for any software, while statistical modeling techniques are applicable only for multi-release software.

The literature contains many SRGM-based modeling techniques and statistical modeling techniques for predicting information about the count or rate of field defects; however, a guide that helps software producers to compare techniques and to select appropriate techniques for their needs is currently unavailable. Prior work by Farr in [55], Khoshgoftaar and Selyia in [45], and Ebert in [14] suggest that at least forty SRGM-based modeling techniques and statistical modeling techniques have been proposed in the literature. However, no prior work has examined the ability of both SRGM-based modeling techniques and statistical modeling techniques to predict information about the count or rate of field defects.

To help software producers, we provide a catalog of techniques that are commonly used in the literature for predicting information about the count or rate of field defects. First, we help software producers understand SRGM-based modeling techniques and statistical modeling techniques by providing a synopsis of the two categories of techniques. Second, we help software producers compare and select techniques by describing individual techniques. Third, we help software producers anticipate techniques that may become commonly used in the future by discussing three promising techniques that address some of the problems with the techniques that are commonly used today.

Our catalog of techniques uses a novel schema to present information on how to use the techniques and information on actual uses of the techniques in practice. Since users of SRGM-based modeling techniques and statistical modeling techniques need to collect inputs, construct models, and then make predictions, we include information on the intuition behind each technique, and its inputs, outputs, and procedures (i.e., how the outputs are produced using the inputs). We incorporate information in previous surveys on SRGM-based modeling techniques, such as [63] by Musa et al., and on statistical modeling techniques, such as [40] by Khoshgoftaar et al. In addition, we also include information on uses of the technique in practice. We include information on applicability, cost of use, and quality of predictions, which software producers are likely to need in order to select appropriate techniques for their needs as suggested by Iannino et al. in [23]. Prior work that compares techniques does not contain information on applicability, cost of use, and quality of predictions. We incorporate information in the literature on uses of the techniques in practice at companies such as IBM [87] and AT&T [71].

In addition to the techniques that we examine in the catalog, many other techniques that predict information about the count or rate of field defects have been published in the literature. Even though we discuss a few promising techniques, we generally exclude techniques that do not have all four characteristics that we discussed above. For example, we exclude:

- Techniques that use expert opinion to make predictions, such as Bayesian belief networks discussed in [65] by Neil and Fenton and the Delphi method discussed in [53] by Linstone and Turoff;
- Techniques that cannot make predictions before the time of release, such as the recalibration using u-plots technique discussed in [6] by Brocklehurst et al.;

- Techniques that have not been used to make predictions for real-world software in prior work, such as the architecture-based technique discussed in [73] by Popstajanova and Trivedi and COQUALMO (COnstructive QUALity MOdel) discussed in [8] by Chulani;
- Techniques that have not been used by people who are not of the group of people that developed the techniques, such as the dynamic weighted linear combination technique discussed in [56] by Lyu and Nikora.

Furthermore, we do not include techniques that analyze defects but do not *predict* information about the count or rate of field defects, such as Orthogonal Defect Classification and Root Cause Analysis, discussed by Clark and Zubrow in [10]. In addition, since we include techniques that are *commonly* used in the literature as judged by the author based on a survey of the literature, we provide references in Section 3 to resources that discuss additional techniques.

Managing the quality of software in the field is important to producers of software. Since software consumers can often switch to an alternative software product if they are not satisfied with the quality of their current software product, software quality is important to the business success of a software producer [9]. Furthermore, software consumers often report quality problems that software producers must expend resources to fix. For example, software service contracts typically specify that a software producer must resolve a customer reported quality problem within a certain amount of time or face penalties [7]. The NIST estimates that poor software quality costs software producers approximately $21.2 billion each year in repair costs [66].

Although the primary audience of this catalog is software producers, consumers of open source software can also use information in this catalog to predict information about the count or rate of field defects, which can help them evaluate the software for adoption [11]. Since many organizations are electing to use open source software in systems and applications that are critical to the business success of the organizations, as discussed in [58] by Mockus et al., the organizations may want to expend resources to evaluate candidate software. Software consumers can usually obtain the inputs needed by the techniques in this catalog for open source software as shown by Li et al. in [48].

This catalog serves two primary purposes. First, it helps software producers manage the quality of their software products by helping them select techniques to use to predict information about the count or rate of field defects, which is often needed by software producers in order to carryout quality management activities. Second, it supports the predictive analysis of design (PAD) framework [82]. The techniques that we examine aid the evaluation of designs prior to adoption and fit within the PAD framework, discussed by Shaw et al. in [82], as predictor functions. This catalog demonstrates that the PAD framework can describe predictive techniques that are used in practice. The techniques that we examine predict information about the count or rate of field defects, which is an implementation property. The techniques use information on the design, the development method, and/or the implementation to make predictions.

Section 2 discusses the common schema that we use to describe the techniques that we examine. Section 3 gives a synopsis of the techniques. Section 4 presents the catalog of techniques. Section 5 discusses promising research. Section 6 summarizes this catalog.

2. Description of the common schema

We adapt the schema used to describe predictor functions in the PAD inventory of predictive techniques [77] to describe the techniques that we examine. However, since we have already discussed how the techniques fit within the PAD framework, we do not include that information in the descriptions. The structure of the schema is:

- Header – states the name and primary output of the technique;
- Abstract – summarizes the purpose, kind, model, and cost of the technique in succinct form;
- Overview – provides more details, including:
    - Inputs – lists information required by the technique,
    - Outputs – lists information generated by the technique,
    - Model – describes the model underlying the technique,
    - Applicability – identifies key constraints on where the technique can be used;
- Procedures – describes the series of successive bindings of inputs within the technique;
- Cost of use – discusses effort related to applying each procedure within the technique;
- Quality of predictions – discusses accuracy of the outputs;
- Related techniques – lists techniques that extend or modify this one;
- References – cites sources that describe aspects of the technique in more detail;

3. Synopsis of the techniques

Many of the techniques that we examine share the same overall approach. This section presents information that can be generalized about the techniques. Information that is specific to individual techniques is in Section 4.

3.1 Header

Field defect is intended to be generic and to encompass all the terms that are used in the literature to describe software related problems that occur after release, such as error [88], fault [63], failure [33], bug [69], and defect [18]. The techniques that we examine are not specific to a particular definition; however, when we discuss prior work that has used a technique, we use the terminology used by the authors of the prior work.

Three kinds of information about the count or rate of field defects are commonly predicted in the literature and are described below. We provide examples of how each kind of information can help software producers with the process of allocating resources for maintenance.

- The field defect rate: the field defect count in each time interval after release.
    - For example, if the time interval is one month, then information on the field defect rate can help software producers evaluate resources needed each month.
- The field defect count: the count of field defects in one time interval.
    - For example, if the time interval is the first year after release, then information on the field defect count can help software producers evaluate the total amount of resources needed that year.
- The field defect thresholding: the field defect count is or is not above a pre-set threshold. Thresholding is a special case of the broader concept of classification; however, we use the term thresholding because prior work usually only considers two classes, that is, either the field defect count is below a pre-determined threshold or it is above the threshold.
    - For example, if the threshold is zero, then information on the field defect thresholding can help software producers evaluate if they will need to allocate resources to deal with field defects.

3.2 Abstract

The techniques that we examine are empirical modeling techniques. They are different from empirical models, such as the generic COCOMO II model for predicting the effort and time to implement a software product [3]. For the COCOMO II model, users of the model collect the inputs and then use the pre-constructed model to make predictions. For the techniques that we examine, users of the technique collect the inputs, construct the models, and then make predictions.

3.3 Overview

3.3.1 Inputs

The techniques that we examine use software metrics as inputs. Software metrics are measures of attributes of the software and are discussed in more detail by Fenton and Pfleeger in [16]. We briefly discuss the software metrics that are commonly used in the literature to make predictions and how to collect them in Appendix A.

SRGM-based modeling techniques

SRGM-based modeling techniques usually use one of two software process metrics that measure development defects to make predictions: the occurrence time of each development defect or the defect count in each time interval during development.

Statistical modeling techniques

Statistical modeling techniques usually use information on field defects from prior releases, software metrics from prior releases, and software metrics from the current release to make predictions. Most statistical modeling techniques can use software metrics that measure various attributes of the software to make predictions. A discussion of how to select the appropriate software metrics to use is in Appendix A.

3.3.2 Outputs
The techniques that we examine predict the field defect rate, the field defect count, and/or the field defect thresholding at the systems level, that is, for the software as a whole. We focus on making predictions at the systems level because software producers generally view the software as a whole. However, prior work also makes predictions for files and modules.

SRGM-based modeling techniques
Prior work has used SRGM-based modeling techniques to predict the field defect rate and the field defect count.

Previous studies that use SRGM-based modeling techniques generally make predictions for the entire software. However, it is likely that the techniques can also be used to make predictions for individual modules as shown by Laprie et al. in [47].

Statistical modeling techniques
Prior work has used statistical modeling techniques to predict the field defect count and the field defect thresholding.

Previous studies that use statistical modeling techniques usually make predictions for modules; however, the techniques should scale up. Users of the techniques should be able to produce predictions for the entire software because prior work has produced predictions for files, such as in [69] by Ostrand et al., and several files constitute a module or component in the same way that several modules constitute a software product. Furthermore, users of the techniques should be able to combine predictions for modules to produce predictions for the entire software as discussed in [88] by Yamada and Osaki.

3.3.3 Models
The theories behind SRGM-based modeling techniques and statistical modeling techniques are different, that is, the justifications for their validity are different. SRGM-based modeling techniques are based on the theory that the occurrence of defects follows some underlying probability function that varies with time. Statistical modeling techniques are based on the theory that some characteristics of the software are related to the occurrence of field defects.

SRGM-based modeling techniques
SRGM-based modeling techniques are based on the theory that the probability of a defect occurrence changes over time as defects are discovered and removed [63]. A SRGM is a mathematical function of time that captures this changing probability. SRGM-based modeling techniques assume that the defect pattern, i.e. the defect count in each time interval, can be modeled using SRGMs. SRGM-based modeling techniques fit SRGMs using development defect information and then make predictions for future time intervals using the fitted SRGMs.

SRGM-based modeling techniques are further divided into two sub-categories: finite and infinite. Finite SRGM-based modeling techniques assume that the total count of field defects that are expected to be discovered is finite. This could be due to reliability growth of the software or user migration to other software (or newer releases of the same software), as discussed by Jones and Vouk in [55]. The Exponential modeling technique, discussed in Section 4.2, is an example of a finite SRGM-based modeling technique. Infinite SRGM-based modeling techniques assume that the total count of field defects that are expected to be discovered is infinite. This could be due to imperfect repair of defects, as discussed by Musa et al. in [63]. The Logarithmic modeling technique, discussed in Section 4.4, is an example of an infinite SRGM-based modeling technique. Infinite SRGM-based modeling techniques are usually not used to predict the field defect count, since the total number of field defects is assumed to be infinite.

For SRGM-based modeling techniques, the independent variable in the constructed model is usually the value of the time interval, and the dependent variable is usually the field defect count in the time interval.

Statistical modeling techniques

Statistical modeling techniques are based on the theory that some attributes of the software are related to the occurrence of field defects [79]. Information on these attributes is captured using software metrics. Software metrics that measure the product, such as lines of code, and the (development) process, such as the number of development defects, are commonly used in the literature to make predictions and are discussed in Appendix A. Statistical modeling techniques assume that the software metrics used to construct models are related to field defects. Statistical modeling techniques build statistical models using information on field defects and software metrics from historical releases and then make predictions using the constructed model and software metrics for the new release.

Statistical modeling techniques are further divided into two sub-categories: parametric and non-parametric. Parametric statistical modeling techniques assume that the relationships between characteristics of the software and field defects occurrences have some structural form. For example, the Linear regression modeling technique, discussed in Section 4.6, assumes linear relationships between software metrics and field defect occurrences. Different parametric statistical modeling techniques assume different structural forms. Non-parametric statistical modeling techniques do not assume that the relationships between characteristics of the software and field defect occurrences have structural forms. For example, the Trees modeling technique, discussed in Section 4.7, assumes that similar historical releases have similar field defect occurrences. Different non-parametric statistical modeling techniques differ in how they decide which historical releases are similar.

For statistical modeling techniques, the independent variables in the constructed models are usually the software metrics and the dependent variable is usually the field defect count or the likelihood of the field defect thresholding.

3.3.4 Applicability

Applicability of the techniques that we examine is related to the assumptions that they make. If an assumption made by a technique is violated in a particular setting, then users of the technique may not be able to use the technique to make predictions or the predictions made by the technique may not be as accurate as predictions made in other settings where the assumption holds [55]. These assumptions are different from the assumptions required to obtain the inputs for the techniques, which are discussed in Appendix A.

The literature provides little information on the settings in which a technique is *not* applicable. Therefore, in this paper, for each technique, we list the assumptions made by the technique and describe settings where prior work has used the technique to make predictions, that is, where the technique is applicable.

One assumption that is common to all the techniques that we examine is that the model constructed using a technique is used to make predictions for the same software as the software from which the data used to construct the model came from. If this assumption does not hold, then predictions made by the constructed models may not be accurate.

SRGM-based modeling techniques

In addition to the common assumption, SRGM-based modeling techniques make three groups of assumptions:

- They assume that the defect pattern can be modeled using SRGMs, which leads to two further assumptions: each defect has the same probability of occurring and the defects occur independently of each other. If these assumptions do not hold, then users of the techniques may not be able to construct the model or the predictions made by the constructed model may not be accurate.
- They assume that the defect pattern is decreasing at the time of prediction, that is, there is reliability growth. This assumption ensures that it is mathematically possible to construct SRGMs. If this assumption does not hold, then users of the techniques will not be able to construct the model.
- They assume that the software is to be operated in a manner similar to that in which the predictions are to be made, that is, the deployment and development environments are similar and the amounts and kinds of usage during testing are similar to the amounts and kinds of usage in the field. This assumption is the basis for extending the defect pattern described by a model fitted to development defect information to future time intervals. If this assumption does not hold, then predictions made by the constructed model may not be accurate.

Assumptions above and the common assumption are common to the SRGM-based modeling techniques that we examine. We will refer to them as *standard applicability restrictions for SRGM-based modeling techniques* in the descriptions of the individual SRGM-based modeling techniques. Furthermore, finite SRGM-based modeling techniques assume that the total count of field defects that are expected to be discovered is finite and infinite techniques assume that the total count is infinite. If these assumptions do not hold then the constructed finite and infinite SRGMs models may not produce accurate predictions. We will refer to these as *standard applicability restriction for finite SRGM-based modeling techniques* and *standard applicability restriction for infinite SRGM-based modeling techniques* in descriptions of the individual techniques. Refer to Lyu [55] and Musa et al. [63] for details about these assumptions.

SRGM-based modeling techniques are generally applicable for any software since they use only development defect information to make predictions. Prior work has used SRGM-based modeling techniques to make predictions for custom-built software, such as a military command and control systems examined in [62] by Musa, and commercial software systems, such as an IBM application system examined in [30] by Kan. However, Li et al. have found that it is not possible to use several commonly used SRGM-based modeling techniques to make predictions for an open source software in [48], because the rate of defects was not decreasing at the time of release.

Statistical modeling techniques

In addition to the common assumption, statistical modeling techniques make three assumptions:

- They assume that the same software metrics used to construct the model are used to make predictions. If this assumption does not hold, then users of the technique may not be able to make predictions using the constructed model.
- They assume that the software metrics used in the model capture sufficient information on attributes of the software that are related to field defects to produce accurate predictions. If this assumption does not hold, then the predictions made by the constructed model may not be accurate.
- They assume that historical information on software metrics and field defects is available from at least one historical release. If this assumption does not hold, then it is not possible to construct models.

Assumptions above and the common assumption are common to the statistical modeling techniques that we examine. We will refer to them as *standard applicability restrictions for statistical modeling techniques* in descriptions of the individual statistical modeling techniques. Refer to Hastie [21] for details about these assumptions.

For parametric statistical modeling techniques, the number of releases from which historical information is available has to be greater than the number of software metrics used in the models. Furthermore, depending on the variation in the software metrics and field defects, data from more releases may be required. However, as discussed in Section 3.2.2, users of the techniques can divide the software into modules, which increases the amount of information available to construct models, make predictions for the modules, and then aggregate the predictions to obtain the prediction for the entire software product.

Statistical modeling techniques are specific for multi-release software since they use information on field defects and software metrics from historical releases to construct models. Prior work has used statistical modeling techniques to make predictions for custom-built software, such as a military command and control system examined in [41] by Khoshgoftaar et al., and for commercial software, such as a provisioning system examined in [69] by Ostrand et al.

3.4 Procedures

At an abstract level, the modeling techniques that we examine have the same set of procedures. First, in the planning procedure, users of the technique decide what to predict, what techniques to use to make predictions, and what software metrics to use to make predictions. Then, in the setup procedure, the users compute the software metrics. In the model-building procedure, the users construct the model. Finally, in the prediction procedure, the users make predictions using the constructed model.

3.4.1 Procedure 1: Planning procedure

Prospective users of the techniques first need to define field defects for the software product, that is, what exactly is a field defect for the software product, and determine the kinds of information that they want to predict. Then, the users need to decide the techniques that they want to use to make predictions. Finally, after deciding what techniques to use, the users need to decide which software metrics to use to make predictions. Prior work usually executes this procedure once for each software product, discussed by Basili and Weiss in [1] and by Donnelly et al. in [55]; however, organizations sometimes re-evaluate these decisions for each release of multi-release software, discussed by Birk et al. in [2].

SRGM-based modeling techniques

Users of SRGM-based modeling techniques must decide which software process metric that measure development defects to collect: the occurrence time of each development defect or the defect count in each time interval during development. However, these two metrics are usually interchangeable as shown by Lyu in [55]. This procedure of deciding what to predict, what techniques to use to make predictions, and which metric to use to make predictions is the same for the SRGM-based modeling techniques that we examine. We will refer to this procedure as the *standard planning procedure for SRGM-based modeling techniques*, in descriptions of the individual SRGM-based modeling techniques.

Statistical modeling techniques

Users of statistical modeling techniques must decide what software metrics to collect. This procedure of deciding what to predict, what techniques to use to make predictions, and what metrics to use to make predictions is the same for the statistical modeling techniques that we examine. We will refer to this procedure as the *standard planning procedure for statistical modeling techniques*, in descriptions of the individual statistical modeling techniques.

3.4.2 Procedure 2: Setup procedure

In general, software metrics are computed using data that are recorded as a part of everyday development or maintenance activities, which lowers the costs associated with collecting the metrics, as discussed by Mockus et al. in [59]. For example, software process metrics that measure development defects are usually extracted from development defect data that is recorded in the defect tracking systems. Computing software metrics is discussed in appendix A. Prior work usually executes the setup procedure once for each software or once for each release of a multi-release software.

SRGM-based modeling techniques

Users of SRGM-based modeling techniques need to extract one of two software process metrics that measure development defects for each release. This procedure is the same for the SRGM-based modeling techniques that we examine. We will refer to this procedure as the *standard setup procedure for SRGM-based modeling techniques*, in descriptions of the individual SRGM-based modeling techniques.

Statistical modeling techniques

Initially, users of statistical modeling techniques need to extract field defect information and software metrics selected in the planning procedure for historical releases as well as the software metrics for the new release. For subsequent releases, usually, only the software metrics for the new release need to be extracted. This procedure is the same for the statistical modeling techniques that we examine. We will refer to this procedure as the *standard setup procedure for statistical modeling techniques*, in descriptions of the individual statistical modeling techniques.

3.4.3 Procedure 3: Model-building procedure

Standard statistical software packages, such as R [74], Splus [83], and SAS [76], are usually used in the literature to construct the models.

In this catalog, we assume that the predictions are made at the time of release; however, the techniques that we examine can also be used to construct models and make predictions earlier in the development process.

SRGM-based modeling techniques

Prior work usually uses non-linear least squares regression or maximum likelihood estimation to fit SRGMs. These two model-fitting routines are found in most statistical software packages. This procedure is the same for the SRGM-based modeling techniques that we examine. We will refer to this procedure as the *standard model-building procedure for SRGM-based modeling techniques*, in descriptions of the individual SRGM-based modeling techniques.

Users of the techniques need to execute the model-building procedure for each software product or once for each release of multi-release software. This is because SRGMs are fitted for each software product or software release.

In general, users of SRGM-based modeling techniques can make predictions anytime before the time of release as long as the SRGM can be fitted, as discussed by Musa et al. in [63]. However, the predictions may be inaccurate, since the predictions are based on incomplete development defect information. Users of the techniques can re-construct the model at the time of release to incorporate complete development defect information [63].

Statistical modeling techniques

For many of the statistical modeling techniques that we examine, the procedure to build the model differs; therefore, we discuss this procedure in the descriptions of the individual statistical modeling techniques.

Prior work usually constructs a model and then uses it to make predictions for multiple subsequent releases without updating the model. Khoshgoftaar and Seliya [45] construct a model using information from one release and then use the model to make predictions for the next three releases. Ostrand et al. [69] construct a model using information from two releases and then make predictions for the next ten releases. However, users of the techniques can re-construct the model for each release to incorporate additional data, which can yield a more accurate model, as shown by Karunanithi in [30].

To make predictions before the time of release, users of statistical modeling techniques need to construct models using software metrics that are available at the time of prediction. For example, predictions after completion of the design can be made using software metrics that are available upon completion of the design, as discussed by Khoshgoftaar and Seliya in [43]. However, to use software metrics that are available at the time of release to make predictions, a separate model has to be constructed.

3.4.4 Procedure 4: Prediction procedure

Standard statistical software packages are usually used to make predictions in the literature. The prediction procedure needs to be executed once for each software product or once for each release of multi-release software.

SRGM-based modeling techniques

To make predictions, users insert future time interval values into the constructed model to obtain the predicted field defect count for the future time intervals.

This procedure is the same for the SRGM-based modeling techniques that we examine. We will refer to this procedure as the *standard prediction procedure for SRGM-based modeling techniques*, in descriptions of the individual SRGM-based modeling techniques.

Statistical modeling techniques

For many of the statistical modeling techniques that we examine, the prediction procedure differs; therefore, we discuss this procedure in the descriptions of the individual statistical modeling techniques.

3.5 Cost of use

We compare the cost of use of each technique that we examine based on the effort needed to make a prediction, which we estimate using descriptions of the procedures in prior work. The cost of use is usually not discussed in the literature. For purposes of comparison, we assume that all statistical modeling techniques collect the same software metrics. The cost of use can be:

- Higher than typical,
- Typical, or
- Lower than typical.

We consider the cost of use of the Linear regression modeling technique (see Section 4.6), which is the most widely used modeling technique in the literature, as typical. Users of the Linear regression modeling technique need to execute the planning procedure and the setup procedure, and then users use statistical packages to construct the model and to make predictions. The cost of use of techniques like the Neural networks modeling technique (see Section 4.8), which requires additional effort to format the software metrics and to manually select the best model, is higher than typical. The cost of use of techniques like the Exponential modeling technique (see Section 4.2), which requires less effort to execute the setup procedure, is lower than typical.

For multi-release software, the cost of use is higher for the initial use of a technique than for subsequent uses. This is due to two reasons. First, the planning procedure usually needs to be executed only once for each software product. Second, users need to execute one-time tasks to compute software metrics, such as creating programs to extract the data, as discussed by Fuggeta et al. in [18]. In general, these tasks do not need to be repeated for subsequent releases. Birk et al. [2] find that, relative to the first release, the effort required for planning and collecting metrics in subsequent releases required only ~22% of the effort required for the first release.

The cost of use of SRGM-based modeling techniques is usually lower than typical. This is mainly because SRGM-based modeling techniques require less effort than statistical modeling techniques to execute the setup procedure, as discussed below.

3.5.1 Procedure 1: Planning procedure

The effort to execute this procedure is likely to vary depending on the goals of the project and the people involved. For example, metrics collection for a project with 6 people to develop a software development environment required ~103 person-hours to plan [18], while metrics collection for a retail petroleum systems project with 16 people required ~346 person-hours to plan [2]. Both organizations used the GQM approach [1]. We note that the purpose of collecting the software metrics in [18] and [2] is not only to predict information about field defects.

SRGM-based modeling techniques

This procedure is likely to require less effort to execute for SRGM-based modeling techniques compared with statistical modeling techniques, since users of SRGM-based modeling techniques only need to decide which one of two possible software process metrics to collect.

Statistical modeling techniques

This procedure is likely to require more effort to execute for statistical modeling techniques compared with SRGM-based modeling techniques, since users of statistical modeling techniques have more options about what software metrics to collect.

3.5.2 Procedure 2: Setup procedure

The effort required to execute this procedure varies depending on the number and the kinds of software metrics collected.

SRGM-based modeling techniques

This procedure is likely to require less effort to execute for SRGM-based modeling techniques compared with statistical modeling techniques, since users of SRGM-based modeling techniques only need to collect one software metric for each release. Donnelly et al. [55] estimate that this procedure usually takes less than 48 person-hours to execute, if performed continuously throughout the development process, based on experiences at

AT&T. The authors do not discuss weather this effort includes effort needed to execute one-time tasks.

Statistical modeling techniques
This procedure is likely to require more effort to execute for statistical modeling techniques compared with SRGM-based modeling techniques, since users of statistical modeling techniques usually need to collect the field defect metric and the software metrics selected in the planning procedure for historical releases initially. Then, for each subsequent release, users also need to collect the software metrics. This procedure can take between ~46 person-hours to ~125 person-hours to execute for each release, based on experiences using the GQM approach in [18] and in [2]. The authors state that similar effort is needed to collect the metrics for the initial release.

3.5.3 Procedure 3: Model-building procedure
For many of the techniques that we examine, the effort required to build the model differs; therefore, the cost of use of this procedure is discussed in the descriptions of the individual techniques. However, since the execution of this procedure is usually aided by statistical software packages, this procedure may take at most several hours to execute.

3.5.4 Procedure 4: Prediction procedure
For many of the techniques that we examine, the effort required to make predictions differ; therefore, the cost of use of this procedure is discussed in the descriptions of the individual techniques. However, like the model-building procedure, the execution of the prediction procedure is aided by statistical software packages; therefore, this procedure may take at most an hour to execute.

3.6 Quality of predictions
We present the accuracy of predictions reported in prior work for each technique that we examine. However, comparisons of the accuracy of predictions are generally not possible. One major reason is that not enough research has been done to determine how differences in the context, such as differences in the type of software or the style of development, affect accuracy of predictions, discussed by Ohlsson and Runeson in [68].

In the catalog, we focus on accuracy of predictions because it is the most commonly used criterion in the literature for assessing the quality of predictions; however, we note that other criteria have been used in the literature. For example, Khoshgoftaar and Seliya [43] and Ebert [14] evaluate the simplicity of the predictions, that is, how easy it is for users of the technique to identify what predictors are important for making the predictions.

SRGM-based modeling techniques
Accuracy of predictions of SRGM-based modeling techniques can vary significantly between data sets, as discussed by Brocklehurst et al. in [6] and by Lyu and Nikora in [56]. The literature suggests not selecting a SRGM-based modeling technique a-priori. Instead, users of the techniques should construct several SRGMs and then select the best model to use by comparing the goodness of fit to the training data or accuracy of predictions for historical releases [55]. This comparison does not significantly affect the

cost of use of SRGM-based modeling techniques because little additional effort is needed to make such comparisons. The inputs needed by most SRGM-based modeling techniques are the same and tools are available to automate comparisons, discussed in [25] and in [55].

Statistical modeling techniques
Accuracy of predictions using the same technique can vary due to differences in the software metrics used, the amount of historical data used to construct the models, and details that are specific to a modeling technique, such as the variant of the technique used or technique specific tuning parameters, as discussed by Ohlsson and Runeson in [68]. Relative to a baseline set of software metrics and amount of historical data used to construct the model, using additional software metrics that measure different attribute of the software, such as in Jones et al. [29], and/or using more historical data, such as in Karunanithi [31], are likely to result in more accurate predictions. However, model specific details are rarely discussed in the literature. In addition, for predictions of the field defect thresholding, comparisons are usually not appropriate because there is a trade-off between the false positive rate and the false negative rate, which are the two accuracy criteria usually used in the literature to evaluate accuracy of predictions. Khoshgoftaar et al. discuss this issue in [40].

3.7 Related techniques

Since some techniques in the literature are based on the same underlying model, in the catalog, we present the most representative modeling techniques, one for each model, and refer readers to their variants.

In addition, for statistical modeling techniques, prior work sometimes uses principal component analysis to pre-process the software metrics, which we discuss in Appendix B.

3.8 References

Several resources discuss the techniques that we examine in detail and other techniques that are not examined in the catalog. Additional SRGM-based modeling techniques and more information on the techniques that we examine can be found in [55] by Lyu, in [64] by Musa and Okumoto, in [87] by Yamada et al., in [86] by Wood, and in [67] by NIST. Additional statistical modeling techniques and information on statistical modeling techniques that we examine can be found in [40] by Khoshgoftaar et al., in [4] by Briand et al., and in [21] by Hastie et al.

4. Catalog of techniques

This section presents the catalog of techniques; however, before we present the individual techniques, we summarize the techniques, discuss the accuracy criteria for each kind of information predicted, and discuss the systems examined in the prior work that we surveyed.

In table 1, we summarize the techniques that we examine. We present kinds of information predicted by a technique, the category of modeling techniques that it belongs to, cost of use, a research study that has used the technique to make predictions for real-world software, and the page where detailed information on the technique can be found.

Table 1. Summary of techniques

Kinds of information predicted	***Modeling technique***	***Category of modeling techniques***	***Cost of use***	***Research study***	***Page***
Field defect rate and count	Gamma	Finite SRGM-based	Lower than typical	Yamada et al. [87]	23
Field defect rate and count	Exponential	Finite SRGM-based	Lower than typical	Pant [71]	26
Field defect rate and count	Weibull	Finite SRGM-based	Lower than typical	Musa and Okumoto [64]	30
Field defect rate and count	Logarithmic	Infinite SRGM-based	Lower than typical	Musa and Okumoto [64]	33
Field defect rate and count	Power	Infinite SRGM-based	Lower than typical	Lyu and Nikora [56]	36
Field defect count and thresholding	Linear regression	Parametric statistical	Typical	Khoshgoftaar et al. [41]	39
Field defect count and thresholding	Trees	Non-parametric statistical	Higher than typical	Selby and Porter [81]	42
Field defect count and thresholding	Neural networks	Parametric statistical	Higher than typical	Karunanithi [31]	46
Field defect count	Ratios	Parametric statistical	Lower than typical	Jalote [26]	50
Field defect thresholding	Discriminant analysis	Non-parametric statistical	Typical	Ebert [14]	52
Field defect thresholding	Pareto	Non-parametric statistical	Lower than typical	Ostrand et al. [69]	54

The most widely used measures of accuracy in the literature for each kind of information predicted about field defects is below.

- The most widely used measure of accuracy in the literature for field defect rate predictions are the mean relative error (MRE), the residual sum of squares (RSS), and mean square error (MSE). The mean relative error is $\sum_{i=1}^{N}\frac{(\hat{y}_i - y_i)^2}{y_i^2}$, the RSS is $\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$, mean square error is $\frac{\sum_{i=1}^{N}|\hat{y}_i - y_i|^2}{\sum_{i=1}^{N} y_i}$,where y_i is the actual number of field defects in the *i*-th time interval, $\hat{y}_i$ is the predicted number of field defects in the *i*-th time interval, and N is the number of time intervals in the duration of a release.

- The most widely used measure of accuracy in the literature for field defect count predictions is the absolute relative error (ARE). The ARE is: $\frac{|\hat{y}_i - y_i|}{y_i}$.
- The most widely used measures of accuracy in the literature for field defect thresholding predictions are the rate of false positives (Type I error) and the rate of false negatives (Type II error).

We present the systems examined in the prior work that we surveyed in table 2.

Table 2. Summary of systems

System set	***Description***	***Modeling technique(s)***
System set 1	Yamada et al. [87] predicted software errors for an IBM on-line terminal control program written in structured programming macros and basic assembler language.	Gamma Exponential
System set 2	Lyu and Nikora [56] predicted systems test failures and operation failures for three projects at the Rome Air Development Center. At least one of the systems is a real-time command and control system.	Gamma Exponential Logarithmic Power
System set 3	Wood [86] predicted defects found in the first year after release for a software system at Tandem computers.	Gamma Exponential Weibull
System set 4	Pant [71] predicted failures for an AT&T electronic switching system deployed at one test site.	Exponential
System set 5	Goel and Okumoto [19] predicted errors for one module of a real-time system: the Naval Tactical Data System (NTDS).	Exponential
System set 6	Musa and Okumoto [64] predicted systems test failures and operation failures for “15 sets of data on a variety of software systems, such as real time command and control, real time commercial, military, and space systems, with system sizes ranging from small, 5.7K, to large, 2.4M.”	Exponential Weibull Logarithmic Power
System set 7	Kan [30] and Panlilio-Yap [70] predicted defects for IBM Application System 400.	Exponential Weibull
System set 8	Khoshgoftaar et al. [38], [41], [42], and [39] predicted faults during systems integration and test phase and during the first year after deployment for a large military telecommunications system written in Ada. The software system was divided into modules.	Linear regression Neural networks Discriminant analysis
System set 9	Khoshgoftaar et al. [38] and Karunanithi [31] predicted changes due to faults for a commercial medical imagining system written in Pascal and Fortran. Lind and Vairavan provided the data for this system in [52]. The software system was divided into modules.	Linear regression Neural networks
System set 10	Khoshgoftaar et al. [34] predicted changes due to faults for a telecommunications system written in a high-level language similar to Pascal. The software system was divided into modules.	Linear regression Trees Neural networks
System set 11	Khoshgoftaar and Seliya [43] and [45] predicted problems leading to code changes for a telecommunications system written in PROTEL. The software system was divided into modules.	Linear regression Trees

System set	***Description***	***Modeling technique(s)***
System set 12	Khoshgoftaar et al. [35] predicted faults discovered by customers after release for "a very large legacy telecommunications system written in a high level language, and maintained by professional programmers in a large organization". The software system was divided into modules.	Linear regression
System set 13	Jones et al. [29] and Khoshgoftaar et al. [36] predicted faults discovered by customers after release in a very large telecommunications embedded system written in a high-level language with more than 10 million lines of code. The software system was divided into modules.	Linear regression Neural neworks Discriminant analysis
System set 14	Briand et al. [4] predicted errors during system and acceptance testing in a 260 KLOC Ada system at NASA Goddard Space Flight Center. The software system was divided into modules.	Linear regression Trees
System set 15	Selby and Porter [81] predicted faults for a Hughes system with 100,000 lines of code. The software system was divided into modules.	Trees
System set 16	Ebert [14] predicted faults for several similar telecommunications systems with roughly 1 million lines of code. The software systems were divided into modules.	Trees Neural networks
System set 17	Ohlsson and Runeson [68] predicted faults for a real-time telecommunications software system. The software system was divided into modules.	Discriminant analysis
System set 18	Ostrand et al. [69] predicted faults in an inventory system at AT&T. The software system was divided into files.	Pareto
System set 19	Jalote [26] predict defect for "hundreds of projects" at InfoSys.	Ratio

4.1 Gamma modeling technique for predicting the field defect rate and the field defect count

Abstract

The Gamma modeling technique is a finite SRGM-based modeling technique. Prior work uses this technique to fit a SRGM based on the Gamma function using software process metrics that measure development defects and then uses the fitted model to make predictions. The cost of use of this technique is lower than typical.

Overview

Inputs

- The occurrence time of each development defect or the defect count in each time interval during development

Outputs

- The predicted field defect rate

Model

This technique adjusts the α and β model parameters of the SRGM based on the Gamma function so that the SRGM describes the observed development defect information. Two mathematically equivalent forms of the SRGM, which are used to describe the field defect rate and the field defect count, are:

Field defect rate (for the *t*-th time interval) = $\alpha\beta^2 te^{-\beta t}$, and

Field defect count (aggregated from time 0 to time *t*) = $\alpha(1-(1+\beta t)e^{-\beta t})$.

The α parameter roughly determines the scale of the model, and the β parameter roughly determines the shape of the model. Gamma functions, which are used to predict the field defect rate, and Gamma cumulative functions, which are used to predict the field defect count, with sample parameter values are in figures 1 and 2.

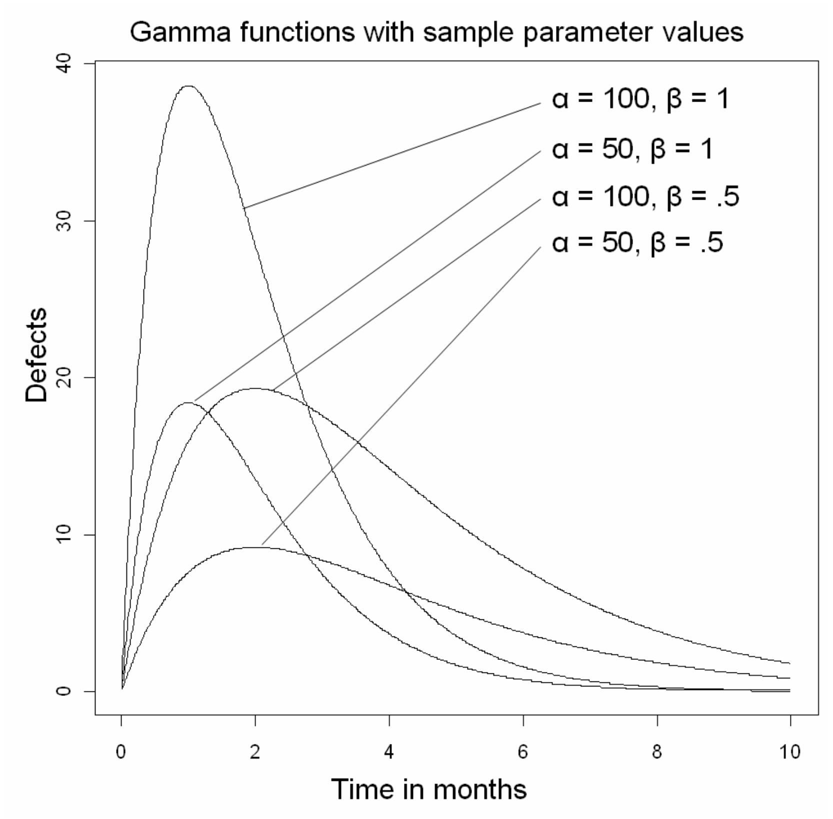

Figure 1. Gamma functions with sample parameter values

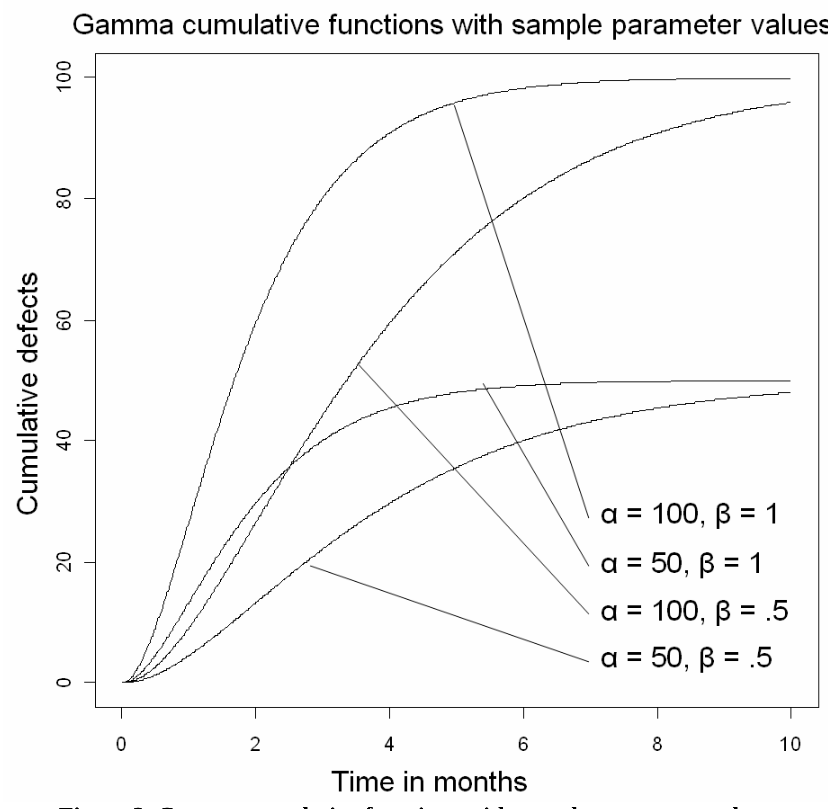

Figure 2. Gamma cumulative functions with sample parameter values

Applicability
This technique has the *standard applicability restrictions for SRGM-based modeling techniques* and *the standard applicability restriction for finite SRGM-based modeling techniques*, discussed in Section 3.3.4. In addition, this technique assumes that the defect pattern can be modeled using the Gamma function.

This technique was used to make predictions for System Set 1, System Set 2, and System Set 3.

Procedures
Users of the technique need to execute the *standard planning, setup, model-building, and prediction procedures for SRGM-based modeling techniques*. These procedures are described in Section 3.4.

Cost of Use
The cost of use of the Gamma modeling technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions
Yamada et al. find in [87] that the RSS for prediction of the error rate for the fit data is 12.6 for 31 errors and the ARE for prediction of the error count is .097 for 41 errors.

Lyu and Nikora find in [56] that the MSE for predictions of the failure rate are 567.7 for ~95 failures for system 1, 246.1 for ~60 failures for system 2, and 2067 for ~145 failures for system 3. For each system, the authors appear to have used ~30% of failures to fit the models initially and then made predictions for the remaining failures. The MSE of this technique ranked third among the five techniques examined by the authors.

Wood finds in [86] that the ARE for predictions of the defects was .029 for 34 field defects.

Related techniques in the catalog
We use the version of the Gamma modeling technique presented in Yamada et al. [87]. Their model is commonly referred to as the S-shaped model in the literature. Littlewood and Verrall apply Bayesian principles to the Gamma modeling technique in [54]. Their model is commonly referred to as the Littlewood-Verrall (LV) model. Their variant allows prior information about the model parameters and about how defect discoveries affect the model parameters to be incorporated into the model.

References
Additional information on the Gamma modeling technique can be found in [55] by Lyu.

4.2 Exponential modeling technique for predicting the field defect rate and the field defect count

Abstract

The Exponential modeling technique is a finite SRGM-based modeling technique. Prior work uses this technique to fit a SRGM based on the Exponential function using software process metrics that measure development defects and then uses the fitted model to make predictions. The cost of use of this technique is lower than typical.

Overview

Inputs

- The occurrence time of each development defect or the defect count in each time interval during development

Outputs

- The predicted field defect rate

Model

This technique adjusts the α and β model parameters of the SRGM based on the Exponential function so that the SRGM describes the observed development defect information. Two mathematically equivalent forms of the SRGM that, which are used to describe the field defect rate and the field defect count, are:

Field defect rate (for the *t*-th time interval) = $\alpha\beta t e^{-\beta t}$, and

Field defect count (aggregated from time 0 to time *t*) = $\alpha(1-e^{-\beta t})$.

The α parameter roughly determines the scale of the model, and the β parameter roughly determines the shape of the model. Exponential functions, which are used to predict the field defect rate, and Exponential cumulative functions, which are used to predict the field defect count, with sample parameter values are in figure 3 and 4.

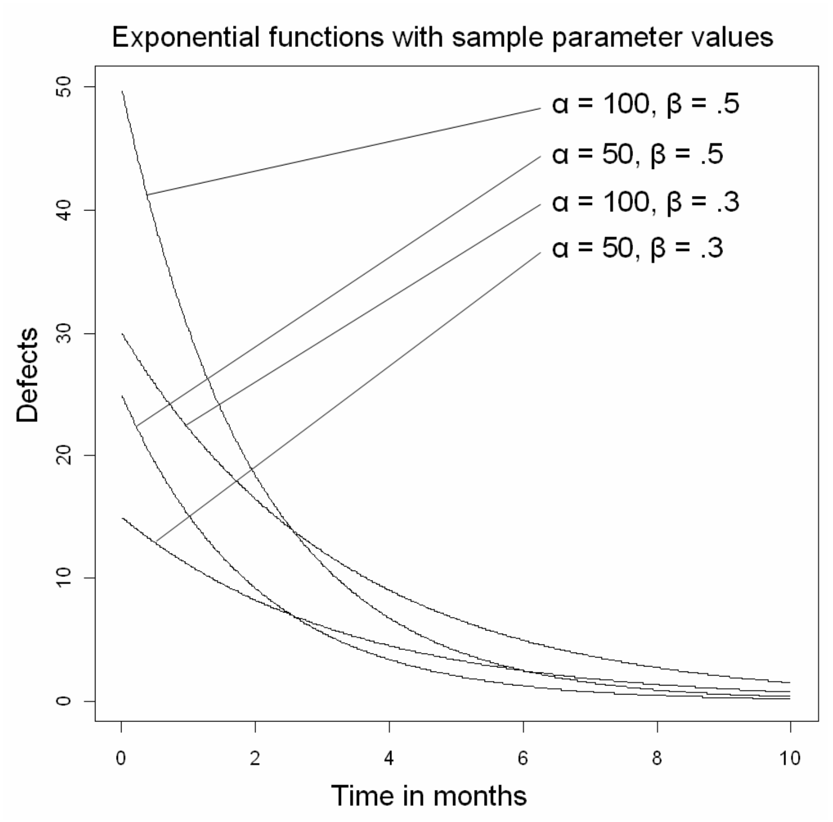

Figure 3. Exponential functions with sample parameter values

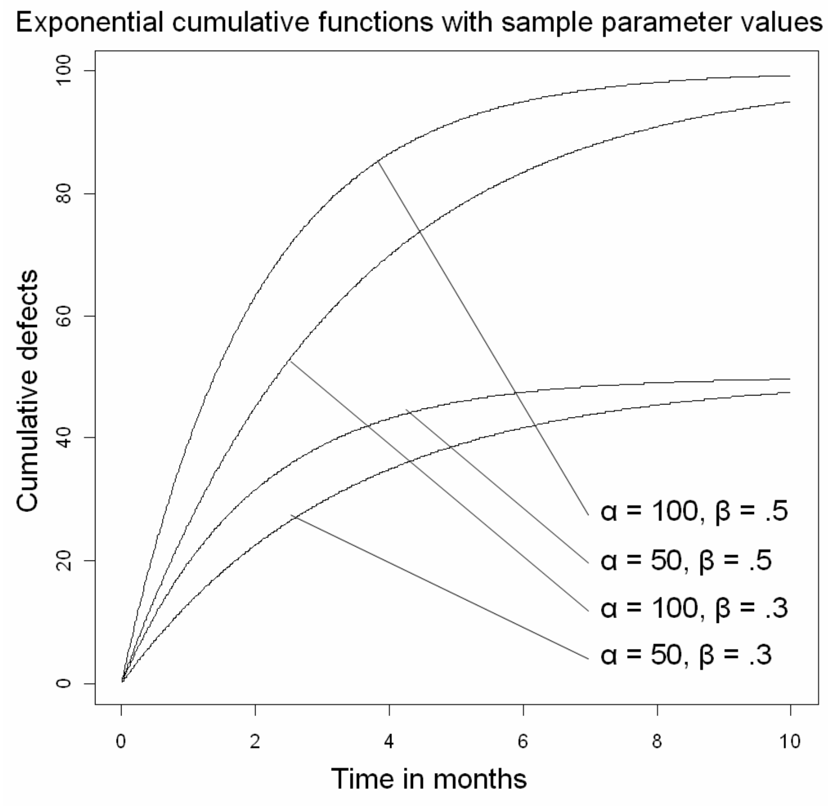

Figure 4. Exponential cumulative functions with sample parameter values

Applicability
This technique has the *standard applicability restrictions for SRGM-based modeling techniques* and *the standard applicability restriction for finite SRGM-based modeling techniques*, discussed in Section 3.3.4. In addition, this technique assumes that the defect pattern can be modeled using the Exponential function.

This technique was used to make predictions for System Set 1, System Set 2, System Set 3, System Set 4, System Set 5, System Set 6, and System Set 7.

Procedures
Users of the technique need to execute the *standard planning, setup, model-building, and prediction procedures for SRGM-based modeling techniques*. These procedures are described in Section 3.4.

Cost of Use
The cost of use of the Exponential modeling technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions
Yamada et al. find in [87] that the RSS for prediction of the error rate for the fit data is 31.5 for 31 errors and the ARE for prediction of the error count is 1.606 for 41 errors.

Lyu and Nikora find in [56] that the MSE for the predictions of the failure rate are 2117 for ~95 failures for system 1, 1455 for ~60 failures for system 2, and 480 for ~145 failures for system 3. Details are in Section 4.1. The MSE of this technique ranked fifth among the five techniques examined by the authors.

Wood finds in [86] that the ARE for predictions of the defects was.029 for 34 field defects.

Pant finds in [71] that “the failure intensity (i.e. the field defect rate) is no more than the value at the time of release thereby validating the measurements made based on verification.”

Goel and Okumoto find in [19] that the 90% confidence bound captures all of the 26 errors used to fit the data, that the ARE of the error count is 0 for 8 post production errors, and that “analyses of the NTDS data and of some other data sets not reported here indicate that the model provides a good fit to the observed failure phenomenon.”

Musa and Okumoto find in [64] that the technique under-estimates the failure rate judged using the median relative error for 15 sets of data.

Kan finds in [30] that the technique is “useful in the development” of the system.

Related techniques in the catalog

We use the version of the Exponential modeling technique presented in Goel and Okumoto [19]. Their model is commonly referred to as the Goel-Okumoto (GO) model in the literature. Musa also proposes this model in [62]. His model is derived slightly differently and is commonly referred to as the Musa basic Exponential model. The Exponential modeling technique is a simplified version of the Weibull modeling technique in Section 4.3. However, the Exponential modeling technique is usually treated as a different modeling technique in the literature.

References

Additional information on the Exponential modeling technique can be found in [55] by Lyu and in [63] by Musa et al.

4.3 Weibull modeling technique for predicting the field defect rate and the field defect count

Abstract

The Weibull modeling technique is a finite SRGM-based modeling technique. Prior work uses this technique to fit a SRGM based on the Weibull function using software process metrics that measure development defects and then uses the fitted model to make predictions. The cost of use of this technique is lower than typical.

Overview

Inputs

- The occurrence time of each development defect or the defect count in each time interval during development

Outputs

- The predicted field defect rate

Model

This technique adjusts the N, α, and β model parameters of the SRGM based on the Weibull function so that the SRGM describes the observed development defect information. Two mathematically equivalent forms of the SRGM that, which are used to describe the field defect rate and the field defect count, are:

Field defect rate (for the *t*-th time interval) = $N\alpha\beta t^{\alpha-1}e^{-\beta t^{\alpha}}$, and

Field defect count (aggregated from time 0 to the time *t*) = $N(1-e^{-\beta t^{\alpha}})$.

The N parameter roughly determines the scale of the model, the α parameter roughly determines the shape of the model, and the β parameter roughly determines the location of the hump in the model. Weibull functions, which are used to predict the field defect rate, and Weibull cumulative functions, which are used to predict the field defect count, with sample parameter values are in figures 5 and 6.

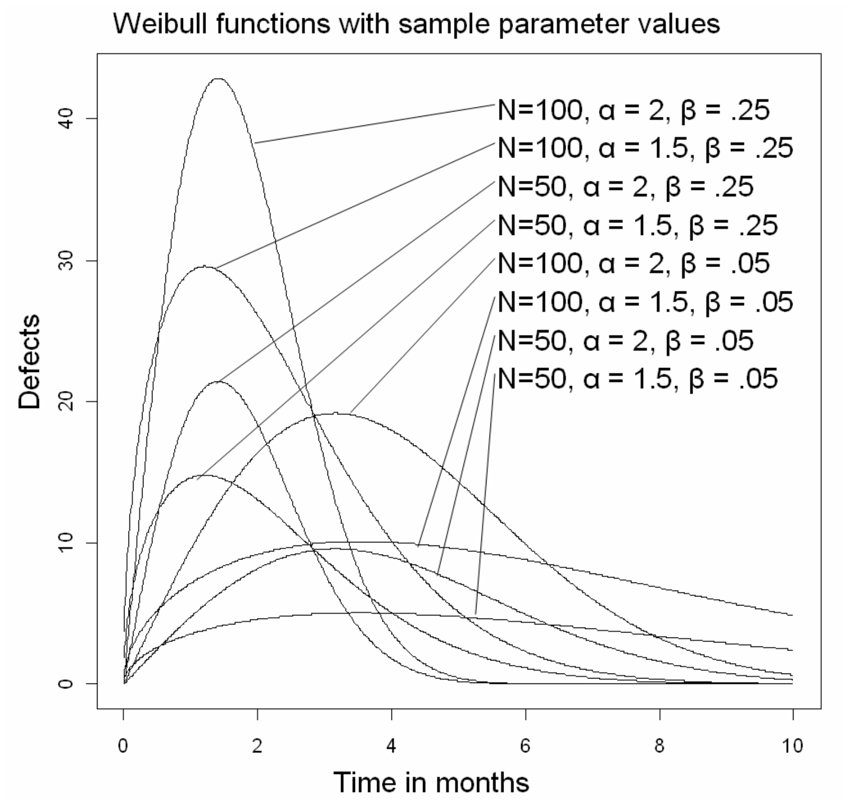

Figure 5. Logarithmic functions with sample parameter values

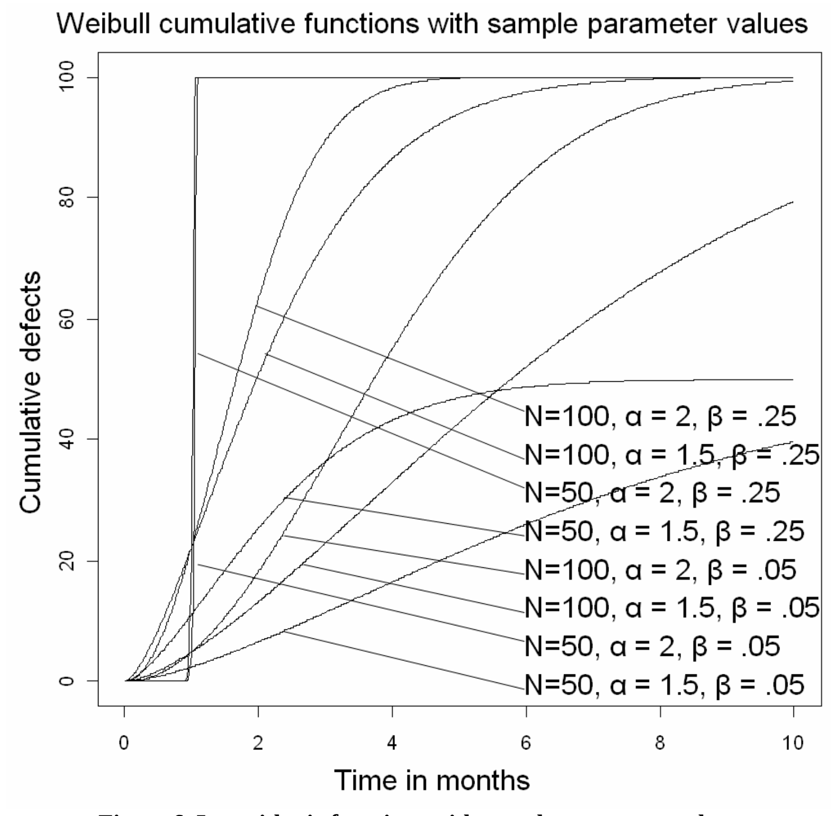

Figure 6. Logarithmic functions with sample parameter values

Applicability
This technique has the *standard applicability restrictions for SRGM-based modeling techniques* and *the standard applicability restriction for finite SRGM-based modeling techniques*, discussed in Section 3.3.4. In addition, this technique assumes that the defect pattern can be modeled using the Weibull function.

This technique was used to make predictions for System Set 3, System Set 6, and System Set 7.

Procedures
Users of the technique need to execute the *standard planning, setup, model-building, and prediction procedures for SRGM-based modeling techniques*. These procedures are described in Section 3.4.

Cost of Use
The cost of use of the Weibull modeling technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions
Musa and Okumoto find in [63] that the Weibull model under-estimates the failure rate judged using the median relative error for 15 sets of data.

Wood finds in [86] that the ARE for predictions of the defects was.029 for 34 field defects.

Kan finds in [30] that the technique is “useful in the development” of the system.

Panlilio-Yap [70] used the technique to model defects for the same system, but the author does not report the accuracy of predictions.

Related techniques in the catalog
We use the version of the Weibull modeling technique presented in Farr [55]. A common variant of the Weibull function is the Raleigh function with is the Weibull function with $\alpha=2$. The Raleigh function is the basis for the Putnam’s Software Life-cycle Model (SLIM) model [70]. SLIM is proprietary and uses different software metrics to construct the model, including an organization’s productivity index and manpower buildup index.

References
Additional information on the Weibull modeling technique can be found in [55] by Lyu.

4.4 Logarithmic modeling technique for predicting the field defect rate

Abstract

The Logarithmic modeling technique is an infinite SRGM-based modeling technique. Prior work uses this technique to fit a SRGM based on the Logarithmic function using software process metrics that measure development defects and then uses the fitted model to make predictions. The cost of use of this technique is lower than typical.

Overview

Inputs

- The occurrence time of each development defect or the defect count in each time interval during development

Outputs

- The predicted field defect rate

Model

This technique adjusts the β_0 and β_1 model parameters of the SRGM based on the Logarithmic function so that the SRGM describes the observed development defect information. Two mathematically equivalent forms of the SRGM that, which are used to describe the field defect rate and the field defect count, are:

Field defect rate (for the *t*-th time interval) = $\frac{\beta_0\beta_1}{\beta_1 t+1}$, and

Field defect count (aggregated from time 0 to the time *t*) = $\beta_0 \ln(\beta_1 t+1)$. (Note that this is an infinite function of *t*).

The β_0 parameter roughly determines the scale of the model and the β_1 parameter roughly determines the shape of the model. Logarithmic functions, which are used to predict the field defect rate, and Logarithmic cumulative functions, which are used to predict the field defect count, with sample parameter values are in figures 7 and 8.

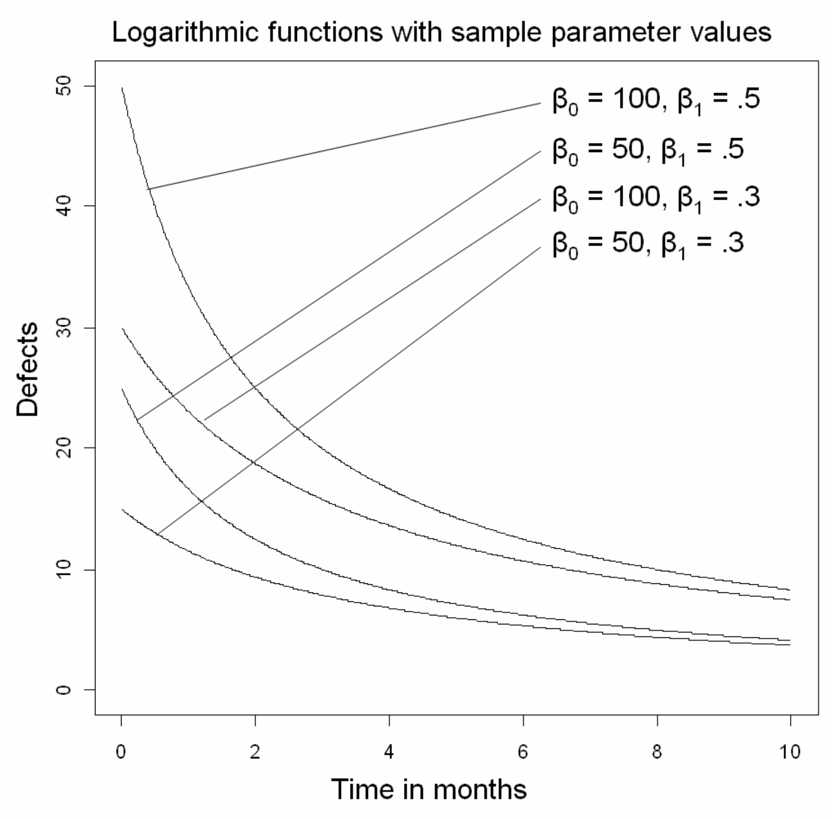

Figure 7. Logarithmic functions with sample parameter values

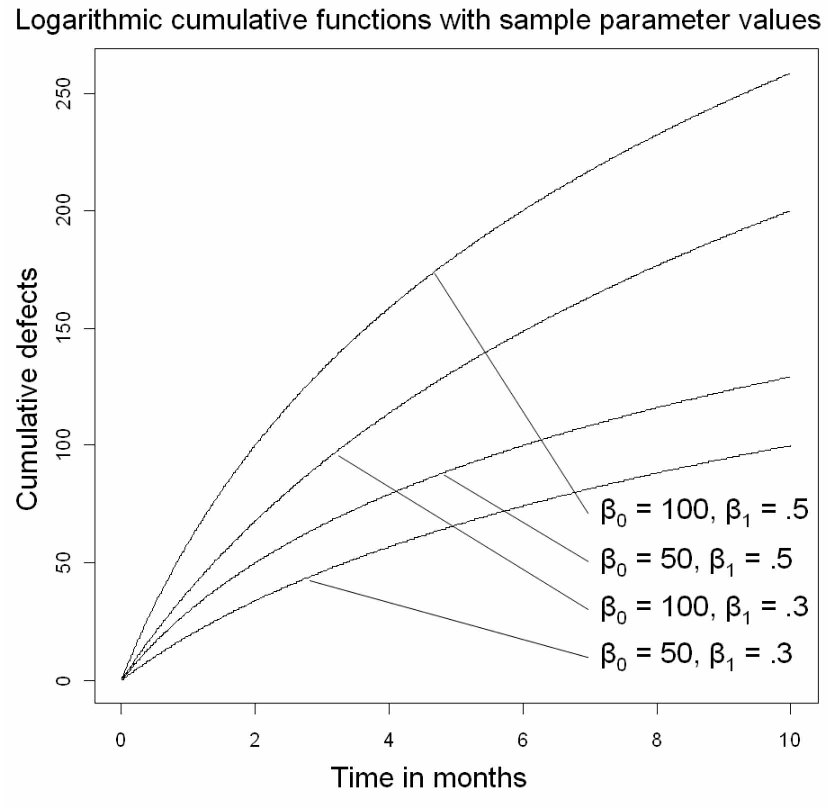

Figure 8. Logarithmic cumulative functions with sample parameter values

Applicability
This technique has the *standard applicability restrictions for SRGM-based modeling techniques* and *the standard applicability restriction for infinite SRGM-based modeling techniques*, discussed in Section 3.3.4. In addition, this technique assumes that the defect pattern can be modeled using the Logarithmic function.

This technique was used to make predictions for System Set 2 and System Set 6.

Procedures
Users of the technique need to execute the *standard planning, setup, model-building, and prediction procedures for SRGM-based modeling techniques*. These procedures are described in Section 3.4.

Cost of Use
The cost of use of the Logarithmic modeling technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions
Lyu and Nikora find in [56] that the MSE for the predictions of the failure rate are 687.4 for ~95 failures for system 1, 1421 for ~60 failures for system 2, and 253.2 for ~145 failures for system 3. Details are in Section 4.1. The MSE of this technique ranked fourth among the five techniques examined by the authors.

Musa and Okumoto find in [63] that the Logarithmic modeling technique is superior to other SRGM-based modeling techniques, including the Exponential modeling technique and the Weibull modeling technique, base on having a better median relative error for predicting fault rates, that is, a median relative error that is closer to zero, for 15 sets of data.

Related techniques in the catalog
We use the version of the Logarithmic modeling technique presented in Musa and Okumoto [63]. Their model is commonly referred to as the Musa-Okumoto (MO) model in the literature.

References
Additional information on the Logarithmic modeling technique can be found in [55] by Lyu and in [63] by Musa et al.

4.5 Power modeling technique for predicting the field defect rate

Abstract

The Power modeling technique is an infinite SRGM-based modeling technique. Prior work uses this technique to fit a SRGM based on the Power function using software process metrics that measure development defects and then uses the fitted model to make predictions. The cost of use of this technique is lower than typical.

Overview

Inputs

- The occurrence time of each development defect or the defect count in each time interval during development

Outputs

- The predicted field defect rate

Model

This technique adjusts the α and β model parameters of the SRGM based on the Power function so that the SRGM describes the observed development defect information. Two mathematically equivalent forms of the SRGM that, which are used to describe the field defect rate and the field defect count, are:

Field defect rate (for the *t*-th time interval) = $\alpha\beta t^{\beta-1}$, and

Field defect count (aggregated from time 0 to the time *t*) = αt^{β}. (Note that this is an infinite function of *t*).

The α parameter roughly determines the scale of the model, and the β parameter roughly determines the shape of the model. Power functions, which are used to predict the field defect rate, and Power cumulative functions, which are used to predict the field defect count, with sample parameter values are in figures 5 and 6.

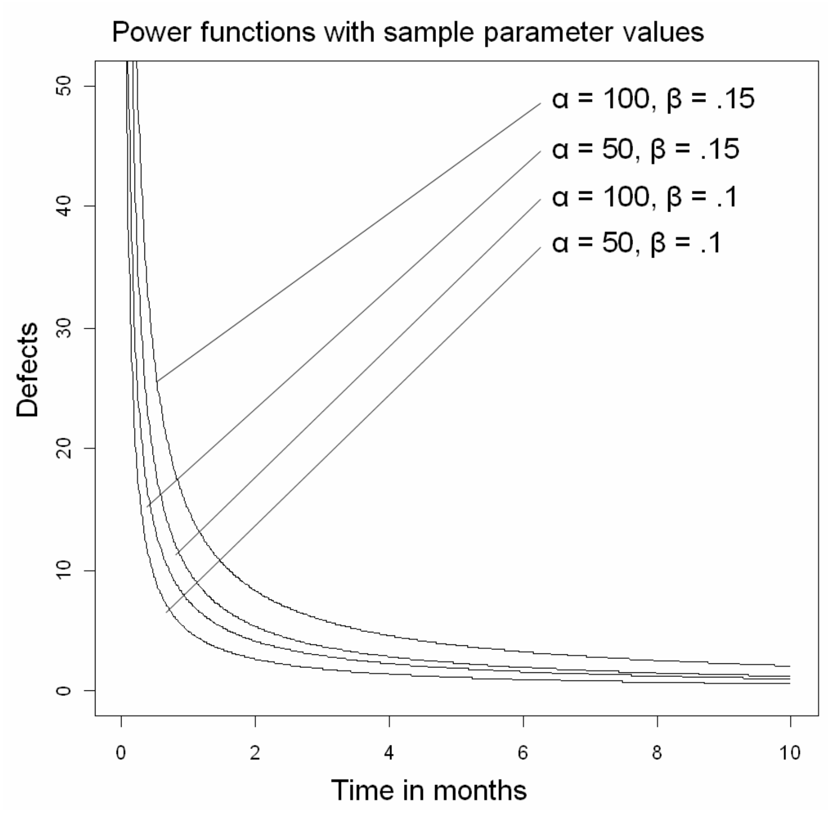

Figure 5. Power functions with sample parameter values

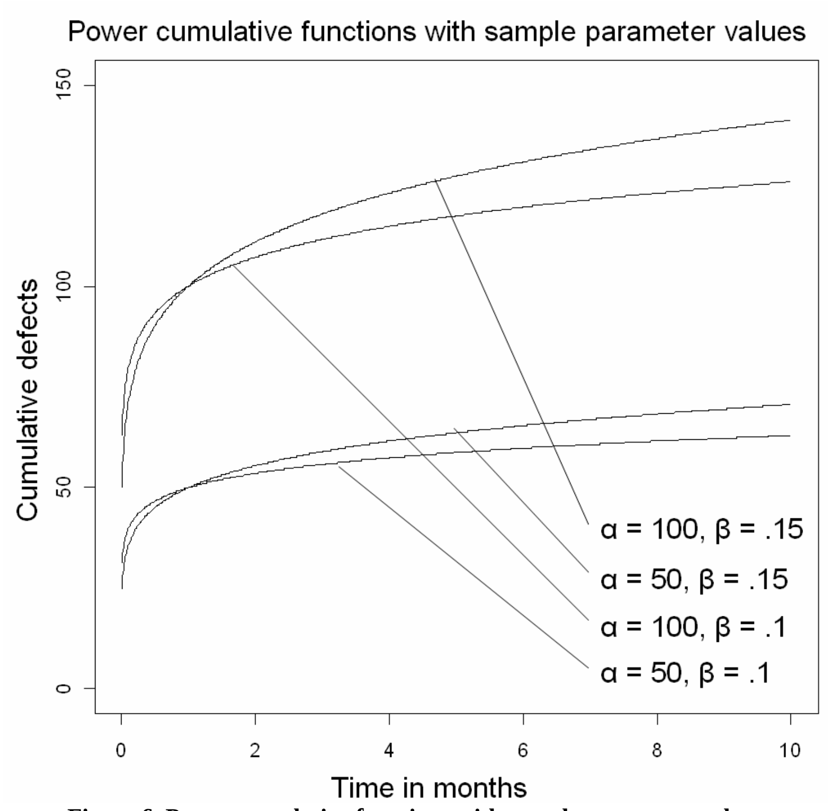

Figure 6. Power cumulative functions with sample parameter values

Applicability
This technique has the *standard applicability restrictions for SRGM-based modeling techniques* and *the standard applicability restriction for infinite SRGM-based modeling techniques*, discussed in Section 3.3.4. In addition, this technique assumes that the defect pattern can be modeled using the Logarithmic function.

This technique was used to make predictions for System Set 2 and System Set 6.

Procedures
Users of the technique need to execute the *standard planning, setup, model-building, and prediction procedures for SRGM-based modeling techniques*. These procedures are described in Section 3.4.

Cost of Use
The cost of use of the Logarithmic modeling technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions
Lyu and Nikora find in [56] that the accuracy of the error rate prediction as measured by the log of the prequential likelihood, which is a measure of the accuracy of predictions based on the probability of experiencing a failure, is -814.3 for ~145 failure, which ranked 8th among the ten techniques that the authors examined. The authors used 60 points, ~30%, of 207 failures to fit the models initially and then made predictions for the remaining failures.

Musa and Okumoto find in [63] that the Power model over-estimates the failure rate judged using the median relative error for 15 sets of data.

Related techniques in the catalog
We use the version of the Power modeling technique presented in Lyu [55]. The model is commonly referred to as the Duane model in the literature. Duane first developed the model at General Electric in 1964, discussed in [13].

References
Additional information on the Power modeling technique can be found in [55] by Lyu.

4.6 Linear regression modeling technique for predicting the field defect count and the field defect thresholding

Abstract

The Linear regression modeling technique is a parametric statistical modeling technique. Prior work uses this technique to fit a Linear model using historical information on software metrics and field defects and then uses software metrics for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is typical.

Overview

Inputs

- Software metrics for historical releases
- Software metrics for the new release
- The field defect count for historical releases

Outputs

- The predicted field defect count or field defect thresholding

Model

This technique uses least squares regression or maximum likelihood estimation to construct a Linear model by adjusting model parameters to fit a Linear model. To predict field defect counts, this technique minimizes the difference between the estimated field defect count and the actual field defect count for historical releases. The Linear model [85] is:

Field defect count = $\beta_0 + \sum_{i=1}^{i=N} \beta_i X_i$, where X_i is value the i-th software metric and N is the total number of software metrics. To predict field defect thresholding, this technique minimizes the difference between the estimated probability field defect thresholding and the actual field defect thresholding: $\log\left(\frac{\hat{p}}{1-\hat{p}}\right) = \beta_0 + \sum_{i=1}^{i=N} \beta_i X_i$, where *p* is the probability that the field defect count is above the threshold.

This modeling technique is usually used in conjunction with model selection, which selects a subset of software metrics to use in the model by examining the change in the goodness of fit resulting from adding or subtracting software metrics from the model [41].

Applicability

This technique has the *standard applicability restrictions for statistical modeling techniques,* discussed in Section 3.4.

This technique was used to make predictions for System Set 8, System Set 9, System Set 10, System Set 11, System Set 12, System Set 13, and System Set 14.

Procedures

Users of the technique need to execute *the standard planning and setup procedures for statistical modeling techniques,* discussed in 3.3.4.

Procedure 3: Model-building procedure

If the user is predicting the field defect thresholding, then the user needs to use the pre-determined threshold to determine the thresholding of historical releases.

(Optional) Use the collected information and a model selection routine, such as backwards elimination used by Khoshgoftaar et al. in [39], found in most statistical software packages to select a subset of the software metrics to use in the model.

Use the model fitting routine found in most statistical software packages to construct the model.

If the user is predicting field defect thresholding, then the user also needs to determine the probability level, that is, the cut-off, at which to classify a release as above the threshold. Prior work usually does this by finding the probability level that balances the Type I and Type II errors in the training set [34].

Procedure 4: Prediction procedure

Insert the software metrics' values for the new release into the constructed model to obtain the predicted field defect count.

If the user is predicting the field defect thresholding, then the user also needs to use the probability level determined in the model-building procedure to decide if the release will be above the threshold.

Cost of Use

The cost of use of this technique is typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure and the prediction procedure in several minutes using standard statistical software packages.

Quality of Predictions

We summarize the accuracy of the field defect count predictions in table 3 and the field defect thresholding predictions in table 4.

Table 3. Accuracy of the field defect count predictions

Study	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*
Khoshgoftaar et al. [41] and [42]	8 software product metrics	188 modules	94 modules	.5877 ARE
Khoshgoftaar et al. [41]	11 software product metrics	226 modules	113 modules	.9998ARE
Khoshgoftaar et al. [34]	9* software product metrics, 2 software process metrics	1320 modules	660 modules	.565 ARE

Study	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*
Khoshgoftaar and Seliya [45], release 2	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3981 modules	.571 ARE
Khoshgoftaar and Seliya [45], release 3	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3541 modules	.602 ARE
Khoshgoftaar and Seliya [45], release 4	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3978 modules	.584 ARE

* Metrics were processed using Principle Component Analysis, see Appendix B

Table 4. Accuracy of the field defect thresholding predictions

Study	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*	*Threshold*
Khoshgoftaar et al. [34]	3 software product metrics, 7 software process metrics, 1 software deployment and usage metric	Not specified	314 modules	27.71% Type I error 22.96% Type II error	0 faults
Jones et al. [29]	24 software product metrics	Half of "a few thousand" modules	Half of "a few thousand" modules	29.06% Type I error 30.77% Type II error	0 faults
Briand et al. [4]	an unspecified number of software code metrics	146 modules, an "equal number of both low- and high-risk" modules	all the high-risk modules and an equivalent number of low risk modules	23.44% Type I error 32.88% Type II error	0 errors

Related techniques in the catalog

We use the Linear regression modeling technique presented in Weisburg [85]. The Linear regression modeling technique is also known as the Multiple regression modeling technique or the Multiple Linear regression modeling technique. Variants of this technique use different measures of accuracy in the model-building algorithm, discussed in Khoshgoftaar et al. [34]. Variants also use different methods to select the software metrics to use in the model, such as in Khoshgoftaar et al. [41]. The version of the Linear modeling technique used to predict the field defect thresholding is also known as the Logistic regression modeling technique.

References

Refer to Weisberg [85] and Khoshgoftaar et al. [41] for details on the Linear regression modeling technique.

4.7 Trees modeling technique for predicting the field defect count and the field defect thresholding

Abstract

The trees modeling technique is a non-parametric statistical modeling technique. Prior work uses this technique to fit a Trees model using historical information on software metrics and field defects and then uses software metrics for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is higher than typical.

Overview

Inputs

- Software metrics for historical releases
- Software metrics for the new release
- The field defect count for historical releases

Outputs

- The predicted field defect count or field defect thresholding

Model

This technique constructs a Trees model by iteratively split historical data into similar groups as judged by deviance of the data in the same node [45]. To predict field defect counts, this techniques measures the deviance of a node *l* as: $\sum_{i \in l} (y_i - u_i)^2$, where y_i is the field defect count of the i-th release and u_i is the mean of the y_i in the same node. To predict the field defect thresholding, this technique measures the deviance of a node *l* as: $1-(p^2(x_1 \mid l)+p^2(x_2 \mid l))$, where $p(x_1|l)$ is the proportion of observations in node *l* that is above the threshold and $p(x_2|l)$ is the proportion of observations in node *l* that is below the threshold.

Each iteration, the tree-building algorithm selects the software metric and metric value that can best split the node into two child nodes that minimizes the sum of the deviance of the left and right child nodes. The splitting finishes when the number of historical releases in the nodes is less than some preset number. The algorithm then prunes the tree using v-fold cross validation (with v usually being 10) to determine the optimal tree.

An example trees model is in figure 7.

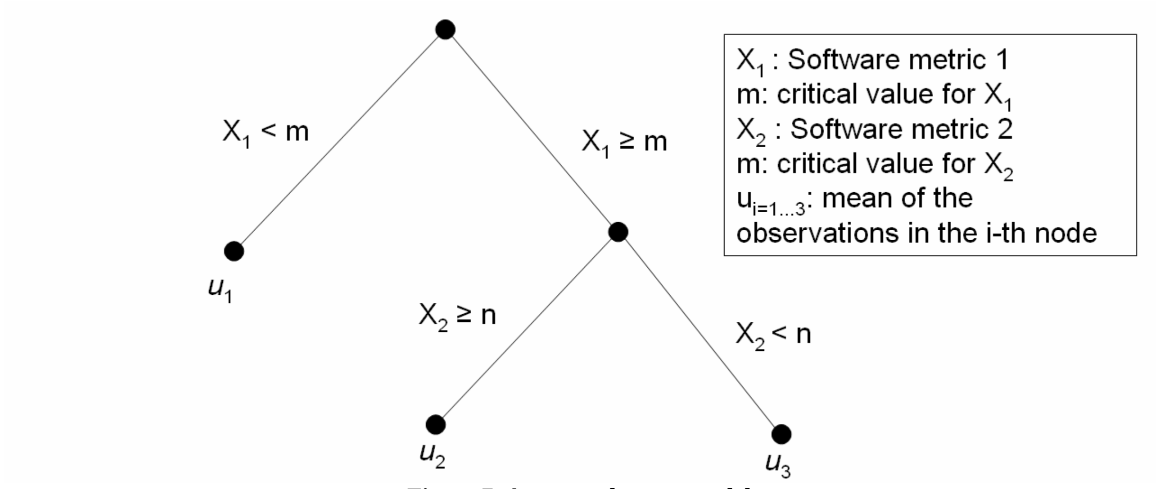
$X_1 < m$
$X_1 \geq m$
X_1 : Software metric 1
m: critical value for X_1
X_2 : Software metric 2
m: critical value for X_2
$u_{i=1...3}$: mean of the observations in the i-th node
u_1
$X_2 \geq n$
$X_2 < n$
u_2
u_3

Figure 7. An example trees model

Applicability

This technique has the *standard applicability restrictions for statistical modeling techniques*, discussed in Section 3.4.

This technique was used make predictions for System Set 10, System Set 11, System Set 14, System Set 15, and System Set 16.

Procedures

Users of the technique first need to execute *the standard planning and setup procedures for statistical modeling techniques*, discussed in 3.3.4.

Procedure 3: Model-building procedure

If the user is predicting the field defect thresholding, then the user needs to use the pre-determined threshold to determine the thresholding of historical releases.

Use the collected information and tree building routine found in most statistical software packages to construct several candidate models by varying the model parameters. Select the candidate model that has the best fit to the historical data.

If the user is predicting the field defect thresholding, then the user also needs to determine the cut-off, that is, the proportion of releases in a leaf node that are above the threshold at which to classify a node as being above the threshold. Prior work usually does this by finding the cut-off that balances the Type I and Type II errors in the training set.

Procedure 4: Prediction procedure

Insert software metrics values for the new release into the constructed model to obtain the predicted field defect count. To make a prediction for a new release, users of the technique traverse the tree based on the software metrics' values of the new release until they reach a leaf node.

If the user is predicting the field defect count, the mean of the field defect counts of the historical releases in the leaf node is the predicted field defect count for the new release.

If the user is predicting the field defect thresholding, then the user needs to use the cut-off to determine if the release will be above the threshold.

Cost of Use

The cost of use of this technique is higher than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure in one person-hour using standard statistical software packages and then execute the prediction procedure in a couple of minutes.

Quality of Predictions

We summarize the accuracy of the field defect count predictions in table 5 and the accuracy of the field defect thresholding predictions in table 6.

Table 5. Accuracy of the field defect count predictions

System	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*
Khoshgoftaar and Seliya [43]	9 software product metrics, 2 software process metric	4648 modules	2324 modules	.3943 ARE
Khoshgoftaar and Seliya [45], release 2	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3981 modules	.324 ARE
Khoshgoftaar and Seliya [45], release 3	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3541 modules	.391 ARE
Khoshgoftaar and Seliya [45], release 4	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3978 modules	.418 ARE

Table 6. Accuracy of the field defect thresholding predictions

System	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*	*Threshold*
Khoshgoftaar and Allen [33], Release 2	24 software product metrics, 14 process metrics, 4 software deployment and usage metrics	"a few thousand" modules from the first release	"a few thousand" modules	27.9% Type I error 28.6% Type II error	0 faults
Khoshgoftaar and Allen [33], Release 3	24 software product metrics, 14 process metrics, 4 software deployment and usage metrics	"a few thousand" modules from the first release	"a few thousand" modules	30.4% Type I error 34.0% Type II error	0 faults

System	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*	*Threshold*
Khoshgoftaar and Allen [33], Release 4	24 software product metrics, 14 process metrics, 4 software deployment and usage metrics	"a few thousand" modules from the first release	"a few thousand" modules	33.7% Type I error 27.2% Type II error	0 faults
Briand et al. [4]	an unspecified number of software code metrics	146 modules, an "equal number of both low- and high-risk" modules	all the high-risk modules and an equivalent number of low risk modules	16.67% Type I error 17.81% Type II error	0 errors
Selby and Porter [81]	2 software process metrics	907 modules was available, information from the first 54 months	907 modules was available. information from the next 12 months	18.84% Type I error 24.32% Type II error	Faults in the top 25% of the training set
Ebert [14]	six complexity metrics	251 modules	200 modules	8.59% Type I error 43.24% Type II error	1 fault

Related techniques in the catalog

We are using the trees modeling technique used in [45] by Khoshgoftaar and Seliya. This technique is also known as the Classification and Regression Trees (CART) modeling technique. Variants of this technique use slightly different measures of deviance for the field defect count, such as $\sum_{i \in l} |y_i - \tilde{y}_i|$, where y_i is the field defect count of the i-th release and $\tilde{y}_i$ is the median of the y_i in the same node. Other variants do not prune the tree and uses an additional parameter to determine when to stop splitting. These variants are discussed in Khoshgoftaar and Seliya [45].

References

Refer to Hastie et al. [21] and Brieman et al. [5] for details on the trees modeling technique.

4.8 Neural networks modeling technique for predicting the field defect count and the field defect thresholding

Abstract

The Neural networks modeling technique is a parametric statistical modeling technique. Prior work uses this technique to fit a Neural networks model using historical information on software metrics and field defects and then uses software metrics for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is higher than typical.

Overview

Inputs

- Software metrics for historical releases
- Software metrics for the new release
- The field defect count for historical releases

Outputs

- The predicted field defect count or field defect thresholding

Model

As explained by Khoshgoftaar et al. in [41], a Neural network is a set of interconnected nodes that have some inputs, an output, and a transformation function. The Neural networks model, arranges the nodes in layers, with one layer for the inputs, one layer for the output, and usually only one intermediate layer, known as a hidden layer. Each node uses its transformation function to compute an output using its inputs. This transformation function is usually a non-linear equation. The input layer has one node for each software metric, and the input to the node is the normalized value of the software metric. Each node in the intermediate layer receives weighted inputs from each node in the input layer. The output layer receives weighted inputs from each node in the intermediate layer and then produces the normalized value of the output.

An example Neural networks model is in figure 8.

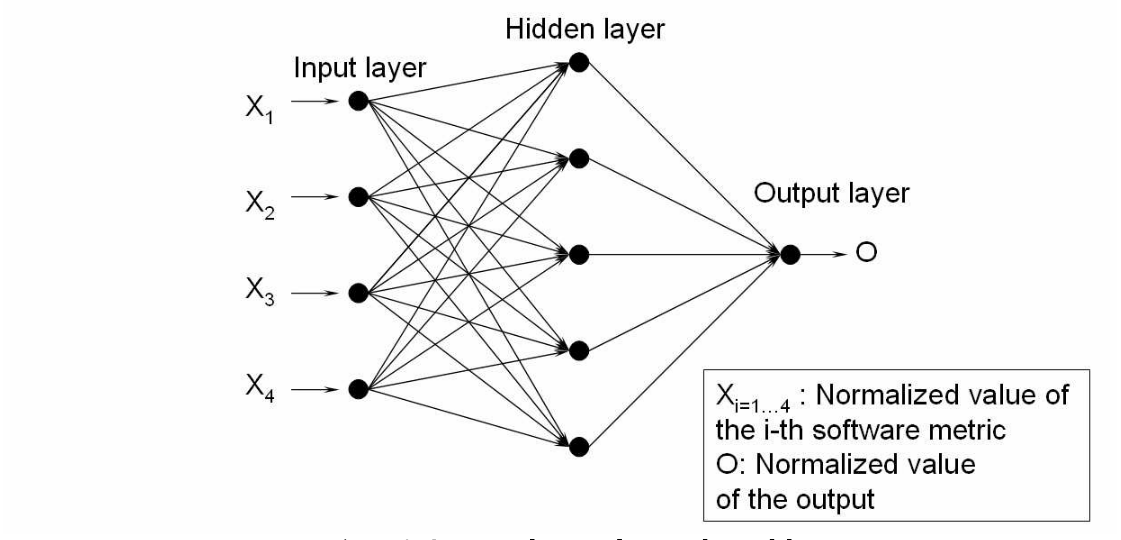
Hidden layer
Input layer
X_1
X_2
X_3
X_4
Output layer
O
$X_{i=1...4}$: Normalized value of the i-th software metric
O: Normalized value of the output

Figure 8. An example Neural networks model

This technique constructs a Neural networks model by adjusting the weights of the inputs and the parameters in the transformation function to fit the observed field defect information. This is usually done using a backwards training algorithm discussed by Khoshgoftaar et al. in [41].

Applicability

This technique has the *standard applicability restrictions for statistical modeling techniques,* discussed in Section 3.4.

This technique was used to make predictions for System Set 8, System Set 9, System Set 10, and System Set 16.

Procedures

Users of the technique first need to execute *the standard planning and setup procedures for statistical modeling techniques,* discussed in 3.3.4.

Procedure 3: Model-building procedure

If the user is predicting the field defect thresholding, then the user needs to use the pre-determined threshold to determine the thresholding of historical releases.

If the user is predicting the field defect count, then the user needs to normalize the field defect count for historical releases by dividing each field defect count by the largest field defect count in historical releases.

Normalize the software metrics by dividing each metric by the largest value of the metric in historical releases. Use the normalized information and the Neural network model fitting routine found in most statistical software packages to construct candidate models by varying the number of intermediate layer nodes. Select the candidate model that has the best fit to historical information. Prior work has constructed candidate models with 5, 10 through 20, 25, and 30 nodes for data sets with 8 software metrics and 11 software metrics, and has found that 16 and 18 intermediate nodes, respectively, had the best fit, discussed in [41] and [42].

If the user is predicting the field defect thresholding, then the user also needs to determine the cut-off of the output at which to classify a node as being above the threshold. Prior work usually does this by finding the cut-off that balances the Type I and Type II errors in the training set.

Procedure 4: Prediction procedure
Normalize the software metrics values for the new release by dividing each software metric by the largest value of the software metric in the training set. Insert the normalized values into the constructed model to obtain the normalized field defect count prediction.

If the user is predicting the field defect count, then the user needs to scales up the output by multiplying the output by the largest field defect count in historical releases to obtain the predicted output.

If the user is predicting the field defect thresholding, then the user uses the cut-off to determine if the release will be above the threshold.

Cost of Use

The cost of use of this technique is higher than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building procedure in several person-hours using standard statistical software packages and then execute the prediction procedure in several minutes.

Quality of Predictions

We summarize the accuracy of the field defect count predictions in table 7 and the accuracy of the field defect thresholding predictions in table 8.

Table 7. Accuracy of the field defect count predictions

Study	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*
Khoshgoftaar et al. [41] and [42]	8 software product metrics	188 modules	94 modules	.3980 ARE
Khoshgoftaar et al. [41]	11 software product metrics	226 modules	113 modules	.5467 ARE
Khoshgoftaar et al. [34]	9 software product metrics, 2 software process metrics	1320 modules	660 modules	.584 ARE
Khoshgoftaar and Seliya [45], release 2	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3981 modules	.620 ARE
Khoshgoftaar and Seliya [45], release 3	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3541 modules	.749 ARE
Khoshgoftaar and Seliya [45], release 4	24 software product metrics, 4 software deployment and usage metrics	3649 modules	3978 modules	.3980 ARE

Table 8. Accuracy of the field defect thresholding predictions

System	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*	*Threshold*
Karunanithi [30]	8 software product metrics	203 modules, after removing modules with between 1-9 faults to improve fitting, trained using 25% of the modules	75% of modules	20.19% Type I error 12.11% Type II error	9 faults
Karunanithi [30]	8 software product metrics	203 modules, after removing modules with between 1-9 faults to improve fitting, trained using 50% of the modules	50% of modules	17.41% Type I error 15.04% Type II error	9 faults
Karunanithi [30]	8 software product metrics	203 modules, after removing modules with between 1-9 faults to improve fitting, trained using 67% of the modules	33% of modules	14.32% Type I error 14.08% Type II error	9 faults
Karunanithi [30]	8 software product metrics	203 modules, after removing modules with between 1-9 faults to improve fitting, trained using 75% of the modules	25% of modules	9.77% Type I error 15.47% Type II error	9 faults
Khoshgoftaar et al. [38]	11 software product metrics	188 module, after removing modules with between 1-4 faults to improve fitting, trained using 75% of the modules	94 modules	12.50% Type I error 6.67% Type II error	4 faults
Ebert [14]	6 software product metrics	251 modules	200 modules	8.64% Type I error 56.76% Type II error	1 fault

Related techniques in the catalog

We are using the Neural networks modeling technique used in [41] by Khoshgoftaar et al.

References

Refer to Hastie et al. [21] for details on the Neural networks modeling technique.

4.9 Ratios modeling technique for predicting the field defect count

Abstract

The Ratios modeling technique is a parametric statistical modeling technique. Prior work uses this technique to fit a Ratios model using historical information on a software metric and field defects and then uses the software metric for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is higher than typical.

Overview

Inputs

- A software metric for historical releases
- A software metric for the new release
- The field defect count for historical releases

Outputs

- The predicted field defect count

Model

This technique computes the ratio of the field defect count to a software metric (e.g. development effort [60]) for historical releases.

Applicability

This technique has the *standard applicability restrictions for statistical modeling techniques,* discussed in Section 3.4.

This technique was used to make predictions for System Set 19.

Procedures

Users of the technique first need to execute *the standard planning and setup procedures for statistical modeling techniques,* discussed in 3.3.4.

Procedure 3: Model-building procedure

Compute the ratio.

Procedure 4: Prediction procedure

Multiply the value of the software metric for the new release by the computed ratio to determine the field defect count for the new release.

Cost of Use

The cost of use of this technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building and prediction procedures in a couple of minutes.

Quality of Predictions

Jalote [26] and Mohapatra and Mohanty [60] report using this technique on “several hundred” projects at Infosys, accuracy of predictions is not reported.

Related techniques in the catalog

We use the Ratios modeling technique used by Mohapatra and Mohanty in[60].

References

Refer to Jalote [26] for details about Ratios modeling technique.

4.10 Discriminant analysis modeling technique for predicting the field defect thresholding

Abstract
The Discriminant analysis modeling technique is a non-parametric statistical modeling technique. Prior work uses this technique to fit a Discriminant analysis model using historical information on software metrics and field defects and then uses software metrics for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is typical.

Overview
Inputs
- Software metrics for historical releases
- Software metrics for the new release
- The field defect count for historical releases

Outputs
- The predicted field defect thresholding

Model
This technique presorts historical releases into classes, that is, a set that is above the threshold and a set that is below the threshold. For each class, the technique computes the probability that a new release belongs to each class using a distance function and a probability function. The distance function used in [38] is: $D_j^2 = (x - x'_j)'\Sigma^{-1}(x - x'_j)$, where *x* is the vector of software metrics for the new release, *x'*$_j$ is the vector of the means of the software metrics in the j-th class, and Σ is the covariance matrix of the software metrics in both classes. The probability that the new releases belongs to the j-th class is: $p_j(x) = \frac{e^{-\frac{1}{2}D_j^2(x)}}{\sum_{i=1}^{2} e^{-\frac{1}{2}D_i^2(x)}}$.

Applicability
This technique has the *standard applicability restrictions for statistical modeling techniques,* discussed in Section 3.4.

This technique was used to make predictions for System Set 8, System Set 13, System Set 16, and System Set 17.

Procedures
Users of the technique first need to execute *the standard planning and setup procedures for statistical modeling techniques,* discussed in 3.3.4.

Procedure 3: Model-building procedure
Place the historical releases into classes.

Procedure 4: Prediction procedure
Use the Discriminant analysis procedure found in most statistical software packages to determine the probability that the new release belongs to each class. Place the new release into the class with the higher probability of class membership.

Cost of Use
The cost of use of this technique is typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique may be able to execute the model-building and prediction procedures in several minutes.

Quality of Predictions
We summarize the accuracy of the field defect thresholding predictions in table 10.

Table 10. Accuracy of field defect thresholding predictions

System	*Metrics used*	*Training set*	*Test set*	*Accuracy of predictions*	*Threshold*
Khoshgoftaar et al. [38]	11 software product metrics	188 module, after removing modules with between 1-4 faults to improve fitting, trained using 75% of the modules	94 modules	20.19% Type I error 12.11% Type II error	9 faults
Ebert [14]	6 software product metrics	251 modules	200 modules	15.95% Type I error 32.43% Type II error	1 fault
Khoshgoftaar et al. [36]	9 software product metrics, 2 software process metrics	1320 modules	660 modules	23.8% Type I error 12.75% Type II error	4 faults
Ohlsson and Runeson [68]	10 software product metrics	28 modules	The same 28 modules	18% Type I error 27% Type II error	10 faults

Related techniques in the catalog
We are using the Discriminant analysis modeling technique used [38] by Khoshgoftaar et al.

References
Refer to Khoshgoftaar et al. [38] for details about the Discriminant analysis modeling technique.

4.11 Pareto modeling technique for predicting the field defect thresholding

Abstract
The Pareto modeling technique is a non-parametric statistical modeling technique. Prior work uses this technique to fit a Pareto model using historical information on software metrics and field defects and then uses software metrics for a new release and the constructed model to make predictions for the new release. The cost of use of this technique is lower than typical.

Overview
Inputs
- Software metrics for historical releases
- Software metrics for the new release
- The field defect count for historical releases

Outputs
- The predicted field defect thresholding

Model
This technique ranks the historical releases based on a software metric. The top 20% of the releases are considered to be above the threshold.

Applicability
This technique has the *standard applicability restrictions for statistical modeling techniques,* discussed in Section 3.4.

This technique was used to make predictions for System Set 16 and System Set 18.

Procedures
Users of the technique first need to execute *the standard planning procedure for statistical modeling techniques,* discussed in 3.3.4.; however, users only need to determine which one software metric to collect. Users then execute the *setup procedures for statistical modeling techniques,* discussed in 3.3.4.

Procedure 3: Model-building procedure
Select a metric to use to rank the releases, and then rank the releases.

Procedure 4: Prediction procedure
Determine the rank of the new release based on the ranking of historical releases. Use the ranking to determine the thresholding of the new release.

Cost of Use
The cost of use of this technique is lower than typical. The cost to execute the planning procedure and the setup procedure is discussed in Section 3.5. Users of this technique

may be able to execute the model-building and prediction procedures in a couple of minutes.

Quality of Predictions

Ostrand et al. [69] report the percentage of defects found in files that are above the threshold, that is files ranked in the top 25% of the files. The authors first fit a linear model using information from two releases and then used the Pareto modeling technique to make predictions for the next 10 releases. The authors used 4 software product metrics and 5 software process metrics to fit the model. The authors find that an average of 80% of the defects is found in the top 25% of the files. The authors then used information from the first 12 releases to fit another linear model. The authors then used the Pareto modeling technique to make predictions for the next five releases. The authors find that an average of 89% of the defects is found in the top 25% of the files. They also examined using just the lines of code metric and information from the first 2 releases to make predictions for the next 15 releases. The authors found that 73% of the defects are found in the top 25% of the files.

Ebert [14] used 6 software product metrics to predict the field defect thresholding. The threshold was 1 fault. Information from 251 modules are used to fit the model and information from 200 modules are used to test the module. The Type I error was 15.95% and the Type II error was 32.43%.

Related techniques in the catalog

We are using the Pareto modeling technique used in [14] by Ebert.

References

Refer to Ebert [14] for details about Pareto modeling technique.

5. Promising research

This section helps software producers anticipate techniques that may become commonly used in the future by discussing three promising techniques that address some of the problems with the techniques that are commonly used today. First, SRGM-based modeling techniques can help software producers to decide whether to conduct more testing before release and to allocate resources for maintenance when the software product is to be operated in a manner similar to that in which the predictions are made, as discussed by Musa et al. in [63] and Lyu in [55]. However, when there are differences between the deployment and development environments and between the amounts and kinds of usage during development and in the field – as is the case for COTS software – prior work has shown that SRGM-based modeling techniques may not produce adequate predictions, such as Kenny in [32] and Li et al in [48]. We examine two lines of research that address this problem:

- Hybrid modeling technique for predicting the field defect rate, and
- Bayesian calibration modeling technique for predicting the field defect rate.

Second, modeling techniques that can identify software metrics that are related to the occurrence of field defects and that can prioritize the software metrics in terms of the strength of the software metrics' relationship to the occurrence of field defects can help software producers by guiding process improvement efforts. However, currently, only the Linear regression modeling technique and the Trees modeling technique are likely to produce models that can help software producers as discussed by Li et al. in [50] and by Selby and Porter in [81]; consequently, software producers may want more choices. We examine a line of research that provides a statistical modeling technique that has both identify-ability and prioritize-ability:

- Boolean Discriminant modeling technique for predicting the field defect thresholding.

We did not discuss these techniques in the catalog because only people who are of the group of people that developed these techniques have used these techniques.

5.1 Hybrid modeling technique for predicting the field defect rate

The hybrid modeling technique combines statistical modeling techniques and SRGM-based modeling techniques. Li et al. [49] uses this technique to construct statistical models to estimate the model parameters of SRGMs that model only field defects using historical information on software metrics and field defect rates. The authors then use software metrics for a new release and the constructed models to predict the field defect rate for the new release. The authors use two Trees models to estimate the two model parameters of the Exponential model to predict the field defect rate in [49].

The hybrid modeling technique removes the assumption that the software product is to be operated in a manner similar to that in which the predictions are to be made by using statistical models to estimate the model parameters of SRGMs. The hybrid modeling technique uses statistical modeling techniques that use historical information on software metrics and field defect rates to determine the relationships between software metrics and the model parameters of SRGMs. Therefore, the constructed models account for differences between the deployment and development environments as well as differences in the usage during development and in the field, as discussed in Section 5.1. In addition, the hybrid modeling technique removes the assumption that the development defect rate is decreasing at the time of prediction by not directly using development defect information to fit SRGMs.

5.2 Bayesian calibration modeling technique for predicting the field defect rate

The Bayesian calibration modeling technique is a SRGM-based modeling technique. Jeske and Akber-Qureshi [28] use this technique to construct SRGMs that model only field defects using historical information on lines of code, development defects, and field defect rates. The authors then use information on lines of code for a new release, development defects for a new release, and the constructed model to predict the field defect rate for the new release. The authors use this technique to estimate the two model parameters of the Exponential model in [28]. The authors use a formula to estimate the model parameter that represents the total number of field defects. The formula estimates the model parameter using information on lines of code added and the effectiveness of testing, which is the ratio of the count of development defects to the count of total defects (both development defects and field defects) for the previous release. The authors then set the model parameter that represents the rate at which field defects are discovered for the previous release as the model parameter for the new release. The authors then apply prior distributions to both model parameters to allow the model parameter to be calibrated using Bayesian methods once field defect data from the new release becomes available.

The Bayesian calibration technique removes the assumption that the software product is to be operated in a manner similar to that in which the predictions are made by using a formula and historical information on actual field defects to estimate model parameters of SRGMs. The formula accounts for differences between the deployment and development environments as well as differences in the amounts and kinds of usage during development and in the field by using data from development and actual field defect data to estimate the model parameter. Similarly, the model parameter that represents the rate at which field defects are discovered also accounts for differences because it is estimated using actual field defect information. In addition, the Bayesian calibration technique removes the assumption that the development defect rate is decreasing at the time of prediction by not directly using development defect information to fit SRGMs.

5.3 Boolean Discriminant modeling technique for predicting the field defect thresholding

The Boolean Discriminant modeling technique is a non-parametric statistical modeling technique. Khoshgoftaar and Seliya [44] use this technique to construct a Boolean Discriminant model using historical information on software metrics and field defect thresholding, and then uses software metrics for a new release and the constructed model to predict the field defect thresholding for the new release. First, the authors rank the software metrics in terms of their Kolomogorov-Smirnov (K-S) test statistic. Second, for each software metric, the authors determine the critical value for the software metric, which is the value of the software metric that has the greatest K-S test statistic. Then, iteratively, the authors add the highest ranked software metric into the model, which identifies a subset of the historical observations as above the threshold. The authors stop when the number of historical observations identified as above the threshold no longer increases with the addition of additional software metrics. For example, assume that "Cyclomatic complexity > X" is the top ranked software metric with critical value X and "Lines of code > Y" is the second highest ranked software metric with critical value Y, where X and Y are constants, then "Cylomatic complexity > X OR Lines of code > Y" is the Boolean Discriminant model using the two highest ranked software metrics. The Boolean Discriminant modeling technique is similar to the Trees modeling technique except that in the Trees modeling technique, the critical values are computed iteratively for each subset of historical observations, as discussed in Section 4.7, whereas in the Boolean Discriminant modeling technique, critical values are computed initially over the set of all historical observations.

The Boolean Discriminant modeling technique is likely to produce models that can help software producers by guiding process improvement efforts because it produces models that has identify-ability and prioritize-ability. The Boolean Discriminant modeling technique identifies the software metrics that are likely to be related to the occurrence of field defects by including only the software metrics that improve the identification of historical releases as above the threshold in the model; furthermore, the technique prioritizes the software metrics used in the model.

6. Summary

Software producers often need information on the rate or count of field defects to perform activities to manage the quality of their software products; therefore, we catalog techniques that are commonly used in the literature to make such predictions. This catalog also shows that the PAD framework [82] can be used to describe predictive techniques that are used in practice because we show how the techniques in this catalog fit within the framework. Hopefully, software producers will use this catalog to better manage the quality of their software products.

7. References

[1] V. Basili and D. Weiss. A Methodology for Collecting Valid Software Engineering Data, In *IEEE Trans. on Software Engineering*, Vol 10, No 6, Nov 1984, pp 728-738.

[2] A. Birk, R. van Solingen, J. Jarvinen. Business impact, benefit, and cost of applying GQM in industry: an in-depth, long-term investigation at Schlumberger RPS. In *Proc. Metrics*, Bethesda, MD, Nov 20-21, 1998, pp 93-96.

[3] B. Boehm et al. *Software Cost Estimation with COCOMO II*, Prentice-Hall 2000.

[4] L. Briand, V. Brasili, C. Hetmanski. Developing interpretable models with optimized set reduction for identifying high-risk software components. In *IEEE Trans. on Software Engineering*, Vol 19, No 11, Nov 1993, pp 1028-1044.

[5] L. Brieman, J. Friedman, R. Olshen, C Stone. *Classification and Regression Trees 2nd Edition.* Wadsworth International Group, 1984.

[6] S. Brocklehurst, P. Chan, B. Littlewood, and J. Snell. Recalibrating Software Reliability Models. In *IEEE Trans on Software Engineering.* Vol 16, No 4, Apr 1990, pp 458- 470.

[7] M. Buckley and R. Chillarege. Discovering Relationships between Service and Customer Satisfaction. *Proc. ICSM*, Opio, France, Oct 17-20, 1995, pp 192-201.

[8] S. Chulani. *Bayesian Analysis of Software Costs and Quality Models*. Ph.D. Dissertation, May 1999. University of Southern California.

[9] S. Chulani, P. Santhanam, D. Moore, B. Leszkowicz, G. Davidson. Deriving a Software Quality View from Customer Satisfaction and Service Data. http://citeseer.ist.psu.edu/chulani01deriving.html, 2001.

[10] B. Clark and D. Zubrow. How good is the software: a review of defect prediction techniques. CMU-SEI. 2001. http://www.sei.cmu.edu/sema/pdf/defect-prediction-techniques.pdf

[11] CMMI product team. CMMI for Development, Version 1.2. *CMU/SEI-2006-TR-008*, 2006.

[12] S. Crawford, A. McIntosh, D. Pregibon. An Analysis of Static Metrics and Faults in C. In *The Journal of Systems and Software,* Vol 5, No 1, Feb 1985, pp 37-48.

[13] J. T. Duane. Learning Curve Approach to Reliability Monitoring. In *IEEE Trans. on Aerospace*. Vol 2, No 2, 1964, pp 563-566.

[14] C. Ebert. Experiences with Criticality Predictions in Software Development. In *Proc. FSE,* Zurich, Switzerland, Sep 22-25, 1997, pp 278-293.

[15] J. Elshoff. Characteristic of Program Complexity Measures. *Proc. ICSE,* Orlando, Fl, Mar 26-29, 1984, pp 188-293.

[16] N. Fenton and S. Pfleeger. *Software Metrics: A Rigorous and Practical Approach, second ed.* PWS, 1997.

[17] N. Fenton and N. Ohlsson. Quantitative Analysis of Faults and Failures in a Complex Software System. In *IEEE Trans. on SWE,* Vol 26, No 8, Aug 2000, pp797-814.

[18] A. Fuggetta, L. Lavazza, S. Morasca, S. Cinti, G. Oldano, F. Orazi. Applying GQM in an industrial software factory. In *TOSEM*. Vol 7, No 4, Oct 1998, pp 411-448.

[19] A. Goel and K Okumoto. Time-dependent Error-Detection Rate Model for Software and Other Performance Measures. In *IEEE Trans. on Reliability*, Vol 28, No 3, Aug 1979, pp 206-211.

[20] S. Gokhale, M. Lyu, K. Trivedi. Analysis of Software Fault Removal Policies Using a Non-Homogeneous Continuous Time Markov Chain. In *Software Quality Journal,* Vol 12, No 3, pp 211-230, 2004.

[21] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning Data Mining, Inference, and Prediction* Springer Series in Statistics 1st ed., 2001.

[22] S. Henry, C. Selig. A Metric Tool for Predicting Source Code Quality from PDL Design. In *Proc. of the Software Design Metrics Workshop*, 1988, pp 28-43.

[23] A. Iannino, J. Musa, K. Okumoto, B. Littlewood. Criteria for Software Reliability Model Comparisons. In *ACM/Sigsoft Software Engineering Notes,* Vol 8, No 3, Jul 1984, pp 12-16.

[24] IEEE standard for a software quality metrics methodology. In *IEEE Std 1061-1998,* 1998.

[25] Information Society Technologies. *Software Reliability Tools.* http://www.ist-pets.com/SRE_tools.htm

[26] P. Jalote. *Software Project Management in Practice.* Addison-Wesley Professional. 2002.

[27] H. Jensen and K Vairavan. An Experimental Study of Software Metrics for Real-Time Software. *IEEE Trans. on SWE*. Vol 11, No 2. Feb 1985, pp 231-234.

[28] D. Jeske, M. Akber-Qureshi. Estimating the failure rate of evolving software systems. In *Proc. ISSRE,* San Jose, CA, Oct 8-11, 2000, pp52-61.

[29] W. Jones, P. Hudepohl, T. Khoshgoftaar, E. Allen. Application of a usage profile in software quality models. In *Proc. 3rd European Conference on Software Maintenance and Reengineering,* Amsterdam, Netherlands, Mar 3-5, 1999, pp148-157.

[30] S. H. Kan. Modeling and software development quality. In IBM Systems Journal, Vol 30, No 3, 1991, pp 351-362.

[31] N. Karunanithi. Identifying Fault-Prone Software Modules Using Feed-Forward Networks: A Case Study. In *Proc. Advances in Neural Information Processing Systems,* Denver, CO, 1993, pp 793-800.

[32] G. Kenny. Estimating Defects in Commercial Software during Operational Use. *IEEE Trans. on Reliability,* Vol 42, No 1, Mar 1993, pp 107-115.

[33] T. Khoshgoftaar, E. Allen. Predicting Fault-Prone Software Modules in Embedded Systems with Classification Trees. In *Proc. the 4th IEEE international Symposium on High-Assurance Systems Engineering,* Washington, DC, Nov 17-19, 1999, pp 105-112.

[34] T. Khoshgoftaar, E. Allen, W. Jones, J. Hudepohl. Return on investment of software quality predictions. In *Proc. IEEE Workshop on application-Specific Software Engineering Technology,* Richardson, TX, Mar 26-28 1998, pp 145-150.

[35] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, N. Goel. Predictive modeling of software quality for very large telecommunications systems. In *Proc. International Conference on Communications,* Dallas, TX, Jun 23-27, 1996, pp 214-219.

[36] T. Khoshgoftaar, E. Allen, K. Kalaichelvan, N. Goel. Early Quality Prediction: A Case Study in Telecommunications. In *IEEE Software.* Vol 13, No 1, Jan 1996, pp 65-71.

[37] T. Khoshgoftaar, E. Allen, A. Naik, W. Jones, J. Hudepohl. Using classification trees for software quality models: lessons learned. In *Proc. International High-Assurance Systems Engineering Symposium*, Washington, DC, Nov 13-14 1998, pp 82-89.

[38] T. Khoshgoftaar, D. Lanning, A. Pandya. A neural network modeling methodology for detection of high-risk programs. In *Proc. ISSRE,* Denver, CO, Nov 3-6, 1993, pp 302-309.

[39] T. Khoshgoftaar, D. Lanning, A Pandya. A comparative study of pattern recognition techniques for quality evaluation of telecommunications software. In *IEEE Journal on Selected Areas in Communications*, Vol 12, No 2, Feb 1994, pp279-291.

[40] T. Khoshgoftaar, J. Munson, D. Lanning. A comparative study of predictive models for program changes during system testing and maintenance. In *Proc. CSM*, Montreal, Canada, Sep 27-30, 1993, pp 72-79.

[41] T. Khoshgoftaar, A. Pandya, D. Lanning. Application of Neural Networks for Predicting Program Faults. In *Annals of Software Engineering*, Vol 1, No 1, Dec 1995, pp 141-154.

[42] T. Khoshgoftaar, A. Pandya, H. More. A neural network approach for predicting software development faults. In *Proc. ISSRE*, RTP, NC, Oct 7-10, 1992, pp 83-89.

[43] T. Khoshgoftaar and N. Seliya. Tree-Based Software Quality Estimation Models for Fault Prediction. In *Proc. Metrics*, Jun 4-7, 2002, pp 203-214.

[44] T. Khoshgoftaar and N. Seliya. Improving usefulness of software quality classification models based on Boolean discriminant functions. In *Proc. ISSRE*, Nov 2002, pp 221-230.

[45] T. Khoshgoftaar and N. Seliya. Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques. In *Empirical Software Engineering*, Vol 8, No 3, Sep 2003, pp 255-283.

[46] T. Khoshgoftaar, R. Shan, E. Allen. Using product, process, and execution metrics to predict fault-prone software modules with classification trees. In *Proc. HASE*, Nov 2000, pp 301-310.

[47] J-C. Laprie, K, Kanoun, C, Beounes, M. Kaaniche . The KAT (knowledge-action-transformation) approach to the modeling and evaluation of reliability and availability growth. In *IEEE Trans. on Software Engineering*, Vol 17, No 4, Apr 1991, pp 370 - 382.

[48] P. Li, J. Herbsleb, M. Shaw. Finding Predictors of Field Defects for Open Source Software Systems in Commonly Available Data Sources: A Case Study of OpenBSD. In *Proc. Metrics,* Como, Italy, Sep 19-22, 2005, pp 22-32.

[49] P. Li, J. Herbsleb, M. Shaw. Forecasting Field Defect Rates Using a Combined Time-Based and Metrics-Based Approach: A Case Study of OpenBSD. In *Proc. ISSRE,* Chicago, Nov 8-11, 2006, pp 193-202.

[50] P. Li, J. Herbsleb, M. Shaw, B. Robinson. Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc. In *Proc. ICSE,* Shanghai, China, May 20-28, 2006, pp 413-422.

[51] H. Li and W. Li. An Empirical Study of Software Metrics. In *IEEE Trans. on Software Engineering,* Vol 13, No 6, Jun 1987, pp 697-708.

[52] R. Lind and K Vairavan. An Experimental Investigation of Software Metrics and Their Relationship to Software Development Effort. In *IEEE Trans. on Software Engineering*, Vol 15, No 5, May 1989, pp 649-653.

[53] H. Linstone and M. Turoff (eds.): *The Delphi Method: Techniques and Applications*, Addison-Wesley, 1975.

[54] B. Littlewood and V. Verrall. A Bayesian Reliability Model with a Stochastically Monotone Failure Rate. In *IEEE Trans. on Reliability*, Vol 23, No 2, Jun 1974, pp 108-114.

[55] M. Lyu. *Handbook of Software Reliability Engineering*. McGraw-Hill, 1996.

[56] M. Lyu and A. Nikora. Applying reliability models more effectively. In *IEEE Software*, Vol 9, No 4, Jul 1992, pp 43-52.

[57] S. Madridakis and S.C. Wheelwright, S.C. *Interactive Forecasting.* Univariate and Multivariate Methods. Second Edition. Holden-Day. San Francisco. 1978.

[58] A. Mockus, R.T. Fielding, J. Herbsleb. Two Case Studies of Open Source Software Development: Apache and Mozilla. In *Trans. on Software Engineering and Methodology*, Vol 11, No 3, Jul 2002, pp 309-346.
[59] A. Mockus, P. Zhang, P. Li. Drivers for Customer Perceived Quality. In *Proc. ICSE*, St. Louis, MO, May 15-21, 2005, pp 225-233.
[60] S. Mohapatra, B. Mohanty. Defect prevention through defect prediction: a case study at Infosys. In *Proc. Software Maintenance*, Florence, Italy, Nov 7-9 2001 pp 260-272 .
[61] J.C. Munson and T.M. Khoshgoftaar. The Dimensionality of Program Complexity. *Proc. ICSE*, Pittsburgh, PA, May 1989, pp 245-253.
[62] J. Musa. A Theory of Software Reliability and Its Application. In *IEEE Trans. on Software Engineering*, Vol 1, No 3, Sep 1975, pp 312-327.
[63] J. Musa. A. Iannino, K. Okumoto. *Software Reliability: Measurement, Prediction, Application.* McGraw-Hill Book Company, 1987.
[64] J. Musa and K. Okumoto. A Logarithmic Poisson Execution Time Model for Software Reliability Measurement. In *Proc. ICSE*, Orlando, FL, Mar 26-29, 1984, pp230-238.
[65] M. Neil and N.E. Fenton, Predicting Software Quality using Bayesian belief networks, In *Proc. of the 21st Annual Software Engineering Workshop*, Washington DC, Dec 4-5 1996, pp 217-230.
[66] NIST. *The Economic Impact of Inadequate Infrastructure for Software Testing.* http://www.nist.gov/director/prog-ofc/report02-3.pdf , 2002.
[67] NIST. NIST/SEMATECH e-Handbook of Statistical Methods, http://www.itl.nist.gov/div898/handbook/, 2006.
[68] M. Ohlsson and P. Runeson. Experience from Replicating Empirical Studies on Prediction Models. In *Proc. Metrics*, Ottawa, Canada, Jun 4-7 2002, pp 217-226.
[69] T. Ostrand, E. Weyuker, T. Bell. Where the Bugs Are. *Proc. ISSTA*, Boston, MA, Jul 11-14 2004, pp 86-96.
[70] Panlilio-Yap. Software Estimation Using the SLIM Tool. In *Proc. Conference of the Centre for Advanced Studies on Collaborative Research*, Toronto, Canada, Nov 9-12, 1992, pp 439-475.
[71] H. Pant. Tracking quality: from verification to customer. In *Proc. GLOBECOM* . Phoenix, AZ, Dec 2-5 1991, pp 158-161.
[72] M. Pighin and A. Marzona. An empirical analysis of fault persistence through software releases. In *Proc. ISESE*, Sep 30-Oct 1 2003, pp 206-212.
[73] K.Popstajanova and K. Trivedi. Architecture based approach to reliability assessment of software systems. In *Performance Evaluation*, Vol 45, No 2-3, Jul 2001, pp 179-204.
[74] R. R Project for Statistical Computing, http://www.r-project.org/
[75] D. Rumelhart, G. Hinton, R. Williams. *Learning Internal Representations by Error Propagation*. MIT Press, Cambridge, MA, 1986.
[76] SAS. *SAS* http://www.sas.com/
[77] C. Scaffidi and M. Shaw. An Inventory of Techniques that Predict Value from Design. In *CMU-ISRI-06-*, 2006.
[78] N. F. Schneidewind. Methodology for Validating Software Metrics. In *IEEE Trans. on Software Engineering*, Vol 18, No 5, May 1992, pp 410-421.
[79] N.E. Schneidewind. Body of knowledge for software quality measurement. In *IEEE Computer*, Vol 35, No 2, Feb 2002, pp 77-83.
[80] A. Schroeder. Integrated Program Measurement and Documentation Tools. *Proc. ICSE*, Orlando, CA, March 26-29, 1984, pp 304-311.
[81] R. Selby and A. Porter. Software metric classification trees help guide the maintenance of large-scale systems. In *Proc. Conference on Software Maintenance*, Miami FL, Oct 16-19, 1989, pp 116-123.
[82] M. Shaw, A. Arora, S. Butler, V. Poladian, C. Scaffidi, In Search of a Unified Theory for Early Predictive Design Evaluation for Software. *Technical Reports CMU-CS-05-139 and CMU-ISRI-05-114*, May 2005.
[83] Splus. *Splus* http://www.insightful.com/
[84] J. Troster and J. Tian. Measurement and Defect Modeling for a Legacy Software System. In *Annals of Software Engineering*, Vol 1, No 1, Dec 1995, pp 95-118.
[85] S. Weisberg. *Linear Regression, 3rd Ed*. Wiley/Interscience, 2005.
[86] A. Wood. Predicting Software Reliability. In *IEEE Computer*, Vol 29, No 11, Nov 1996, pp 69-77.

[87] S. Yamada, M. Ohba, S. Osaki. S-Shaped Reliability Growth Modeling for Software Error Detection. In *IEEE Trans. on Reliability*, Vol 33, No 2, 1983, pp 475-478.
[88] S. Yamada and S. Osaki. Software Reliability Growth Modeling: Models and Assumptions. In *IEEE Trans. on Software Engineering,* Vol 11, No 12, Jan 1985, pp 1431-1437.

Appendix A. Software metrics

The techniques that examine use software metrics as inputs. 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. Software metrics are metrics that measure attributes of a software system.

Current practices for selecting software metrics to collect for producing predictions about the count or rate of field defects are to consider the attributes of the software system that could be related to field defects and then to collect metrics that measure those attributes, discussed by Basilli and Weiss in [1]. This process often involves examining metrics that have been validated in prior work, that is, metrics that have been shown to be statistically associated with field defects, discussed in Scheidewind [78]. However, even a validated metric, such as lines of code, may not be statistically associated with field defects for all systems due to various factors, such as the programming language or the specific definition of “lines of code” used, discussed by Ohlsson and Runeson in [68]. Therefore, the literature recommends focusing on the attribute of the software system being measured, rather than the specific metric used to measure the attribute.

To help practitioners determine what software metrics to collect and how to collect them, we discuss the attribute measured by some commonly used metrics in the literature, the data sources commonly used to collect the metric, the procedures commonly used to collect the metric, and the cost of collection. By commonly used metrics, we mean metrics that are used in multiple studies. In addition, we describe how each metric fits within the PAD framework, that is, whether the metric captures information on the design, the development method, the implementation, or the context.

The two high-level entities that are commonly measured in the literature are discussed below. These entities are discussed in detail by Fenton and Pfleeger in [16], by Khoshgoftaar et al. in [46], and by the IEEE standard for software quality metrics methodology in [24]:

- Software product: metrics that measure this entity measure attributes of any intermediate or final product of the software development process, such as lines of code,
- Software process: metrics that measure this entity measure attributes of the development process, such as the number of development defects.

Information on specific attributes and the software metrics that measure the attributes is in each sub-Section.

Two data sources are commonly used to compute software metrics in prior work:

- Request tracking system: tracks customer reported and developer reported problems, which may not necessarily be software related
- Change management and version control system: tracks changes to the code.

Most of the software metrics used in the literature is collected from these two data sources.

We rate the cost of collection of the software metrics based on the amount of effort needed to collect the metric, which we evaluate subjectively using descriptions of the collection procedures in prior work. The cost of use can be:

- Higher than typical,
- Typical, or
- Lower than typical

The cost of collection of the metric that measures the number of changes to the code (deltas), which is a software development metric (see Appendix A.3), is typical because prior work extracts data on changes to the code from the change management and version control system and then creates programs to compute the number of changes. The cost of collection of the metric that measures the lines of code, which is a software product metric (see Appendix A.2), is lower than typical because prior work usually uses automated tools to compute the lines of code after a snapshot of the code is extracted from the change management and version control system. Using automated tools reduces the amount of effort needed. The cost of collection of the defects during development metric (see Appendix A.2) is higher than typical because prior work usually computes the metric by collecting data from two separate data sources and then creating programs to parse the data and linking the data together. Collecting and parsing data from two sources increases the amount of effort needed.

Appendix A.1 Field defects

Information on field defects is usually computed using data from the request tracking system and the change management and version control system. Prior work usually extracts customer reported problem information from the request tracking system and change information from the change management and version control system. Then, prior work usually creates programs to parse the data based on data fields specific to each software system in order to link the data together and determine which customer reported problems resulted in code changes. The cost of collection of this metric is higher than typical.

In the PAD framework, field defect is a property of the implementation.

Appendix A.2 Software product metrics

The most obvious place to look for attributes of the software system that may be related to field defects is in the software system itself. Software product metrics are the most widely used software metrics in prior work. Many software product metrics have been considered in the literature; however, none is significantly better than lines of code, discussed by Crawford et al. in [12] and by Fenton and Ohlsson in [17].

We describe the attributes measured by the software product metrics that are commonly used in the literature using the descriptions used in Khoshgotaar and Seliya [45] and Munson and Khoshgoftaar [61]. We also consider when the metrics are available. Khoshgoftaar et al. [46] and Troster and Tian [84] identify product metrics that can be computed from design documents before coding starts. This way of categorizing software product metrics is useful when we place software product metrics into the PAD framework. We summarize the software product metrics commonly used in the literature in Appendix table 1.

Appendix table 1. Software product metrics

Attribute measured	***Time of availability***	***Software product metric***	***Data source***	***Collection procedure***	***Cost of collection***
Control flow graph metrics	Post coding	Possible program knot count [51] [45]	Change management and version control system	Prior work usually extracts snapshots of the code and then computes the metrics using automated tools	Lower than typical
		Log of independent paths [45] [15]			
		Number of exit nodes [15] [45]			
	Post design	Cyclomatic complexity [51] [15] [80] [27] [22] [43]			
		Number of loop constructs [45] [43]			
		Number of non-loop conditional arcs [45] [43]			
Statement metrics	Post coding	Unique operand count [51] [15] [80]	Change management and version control system	Prior work usually extracts snapshots of the code and then computes the metrics using automated tools	Lower than typical
		Calculated program length [51] [15] [27]			
		Program vocabulary [51] [15] [80]			
		Total operand count [51] [15]			
		Halstead's program volume [51] [15] [27] [22]			
		Total source statements [51] [80]			
		Total operator count [51] [15]			
		Program length [51] [27] [22]			
		Unique operator count [15] [80]			
		Total source input lines of code [51] [27] [22] [45]			
		Input source code lines [15] [45]			
	Post design	Distinct include files [45] [15]			

Attribute measured	***Time of availability***	***Software product metric***	***Data source***	***Collection procedure***	***Cost of collection***
Degree of modularization of a program	Post coding	Number of call statements [15] [22] [45]	Change management and version control system	Prior work usually extracts snapshots of the code and then computes the metrics using automated tools	Lower than typical
		Mean nesting depth [80] [15]			
		Maximum nesting depth [80]			
	Post design	Number of distinct calls to others [45] [43]			
		Total calls to others [43] [84]			
Mental effort required to generate an implementation from a specification	Post coding	Halstead's program effort [51] [15] [27] [22]	Change management and version control system	Prior work usually extracts snapshots of the code and then computes the metrics using automated tools	Lower than typical

In the PAD framework, software product metrics that can be collected post-design capture properties of the design and software product metrics that can only be collected after coding is completed capture properties of the implementation.

Appendix A.3 Software process metrics

Since the software system is the result of a development process, the next logical place to look for attributes of the software system that may be related to field defects is in the development process. The number of development defects and the number of changes to the code are the two most widely used software process metrics in the literature. Either the occurrence times of development defects or the number of development defects in each time interval during development must be collected in order to use SRGM-based modeling techniques.

We present the software process metrics used in the literature in Appendix table 2. We have inferred the attributes intended to be captured by the metrics based on descriptions of the metrics in the literature.

Appendix table 2. Software process metrics

Group	***Software process metrics***	***Data sources***	***Collection procedure***	***Cost of collection***
Problems discovered prior to release: software process metrics that mention measuring attributes of problems found prior to release in the description	Number of defects identified during the previous release [69] [46]	Request tracking systems, change management and version control systems.	Prior work usually extracts problem report data from the request tracking system and code change information from the change management and version control system. Then, prior work creates programs to parse the data based on data fields specific to each software system to determine which problems resulted in changes to the code	Higher than typical
	The occurrence time of development defects [55] [63]			
	The number of development defects in a time interval [55] [63]			
	Number of development defects [17] [46]			
Changes to the product: software process metrics that mention measuring attributes of changes made to the software product in the description.	Amount of reuse [69] [72] [84] [3] [46]	Change management and version control systems	Prior work usually extracts data on changes to the code and then creates programs to parse the data based on the specifics of the data to determine the kinds and numbers of changes	Typical
	Changes made to a file (deltas) [69] [46]			
	Changed lines of code [84] [46]			
People in the process: software process metrics that mention measuring attributes of people involved in the development process in the description.	Different designers making changes [46] [43]	Change management and version control systems	Prior work usually extracts data on changes to the code and then creates programs to parse the data based on the specifics of the data to obtain information on the people who made the changes	Typical
	Number of updates by designers who had 10 or less total update in their company career [46] [43]			
	Number of updates by designers who had between 11 and 20 total update in their company career [46] [43]			
	Number of updates designers had in their company career [46] [43]			

Group	***Software process metrics***	***Data sources***	***Collection procedure***	***Cost of collection***
Process efficiency: software process metrics that mention measuring attributes of the maturity of the process or the effort in the description.	Design effort [81] [58]	Time sheets	Prior work uses engineer's time sheets to compute effort	Higher than typical
	Coding effort [81] [58]			
	Total effort [81] [58]			

In the PAD framework, software process metrics that measure problems discovered before release and changes to the product capture properties of the implementation. Software process metrics that measure information on people in the process and process efficiency capture properties of the method.

Appendix B. Principal component analysis

Principal component analysis (PCA) constructs variables that are linear combinations of existing variables (i.e. software metrics) to capture most of the information in the original variables while reducing the number of variables, discussed by Khoshgoftaar and Seliya [45]. PCA has been used with many parametric and non-parametric statistical modeling techniques. This is done by first constructing PCA variables and then using the PCA variables in the modeling techniques. PCA increases the cost of use of a technique since additional effort is needed to construct the PCA variables. Some studies have reported increased accuracy using PCA, such as Briand et al. [4], while a study by Khoshgoftaar and Seliya in [45] has reported that differences in accuracy are not statistically significant.