Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Sciences xxx (2006) xxx–xxx
INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL
www.elsevier.com/locate/ins

Identification of defect-prone classes in telecommunication software systems using design metrics

Andrea Janes [a], Marco Scotto [b], Witold Pedrycz [c], Barbara Russo [a], Milorad Stefanovic [c], Giancarlo Succi [a,*]

[a] *Center for Applied Software Engineering, Free University of Bozen/Bolzano, Dominikanerplatz 3 Piazza Domenicani, I-39100 Bozen, Italy*
[b] *Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada*
[c] *DIST—Università di Genova, Italy*

Received 25 November 2004; accepted 6 December 2005

Abstract

The goal of this paper is to investigate the relation between object-oriented design choices and defects in software systems, with focus on a real-time telecommunication domain. The design choices are measured using the widely accepted metrics suite proposed by Chidamber and Kemerer for object oriented languages [S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493].

This paper reports the results of an extensive case study, which strongly reinforces earlier, mainly anecdotal, evidence that design aspects related to communication between classes can be used as indicators of the most defect-prone classes.

Statistical models applicable for the non-normally distributed count data are used, such as Poisson regression, negative binomial regression, and zero-inflated negative

* Corresponding author. Tel.: +39 0471 315 640; fax: +39 0471 315 649.
E-mail address: Giancarlo.Succi@unibz.it (G. Succi).

0020-0255/$ - see front matter
doi:10.1016/j.ins.2005.12.002

binomial regression. The performances of the models are assessed using correlations, dispersion coefficients and Alberg diagrams.

The zero-inflated negative binomial regression model based on response for a class shows the best overall ability to describe the variability of the number of defects in classes.

Keywords: Object-oriented design; Software metrics; Empirical software engineering

1. Introduction

Quality is a key element in the success of any software product. Real-time telecommunication systems have even more stringent requirements for quality, since their failure may result in penalties to developers, losses of essential services for users, and even life threatening situations.

Assuring high quality is an increasingly complex time- and effort-consuming activity. Consequently, it is essential to identify the most critical parts of software systems to properly allocate resources for defect detection and removal. As the cost of fixing defects increases over time [25], it is also important to perform such identification as early as possible.

Moreover, the ability to identify the most defect prone modules early in the development cycle provides critical support for design inspections and for design-based refactoring [14].

Models based on software metrics can be employed towards this goal.

Object-oriented methodologies offer a substantial amount of information about the system even before any coding has started. A widely used set of software metrics for object-oriented systems is the CK suite [9]. The suite has shown its usefulness in several studies [31,3,10,7,42]. However, most of these studies have limited practical effect since they were constructed on data from academic environment, and based on small datasets. In this work, we attempt to review empirically the possibility of early identification of the classes that contain most of the defects in the system.

In particular, using *large industrial datasets*, we validate *for our target application domain* previous anecdotal claims that:

- High values of associations between classes identify defect prone classes.
- Defects are concentrated in a few classes and are distributed according to a Poisson-like curve.

Furthermore, we evidence that *it is possible to recognize defect prone classes early in the software lifecycle*, when defects are easier to fix.

In this paper, we present the results of an extensive survey based on five real-time, telecommunication software systems developed by a North-American company.

For the applications analyzed in this study, all the modifications of the classes caused by defects in software operation were recorded throughout the development process. The number of modifications referring to defects, widely used in other scientific studies [17], represents a good estimation of the defect-proneness of a class.

The internal product metrics are collected from stable product releases, and include the CK metrics suite.

To deal with typical problems related to software metrics, such as absolute measurement scale, non-normal distribution, high variance, and underprediction of zero values, Poisson regression models (PRM) and negative binomial regression models (NBRM) [7] are employed. In addition to these models, we also apply zero-inflated negative binomial regression models (ZINBRM) [33].

The models are evaluated using the correlation between the model estimations and the observed data, and the dispersion parameter. We also use Alberg diagrams for graphical evaluation of models' performances [39,16]. These diagrams represent an additional method for assessment and comparison of the models' ability to identify the most critical classes in the system.

This paper is organized as follows. An overview of the applied metrics and statistical methods used in this study is provided in Section 2. The reference products and the experimental data are presented in Section 3. In Section 4, the regression models are applied to estimate the number of defects. The analysis of the results is presented in Section 5. In Section 6, this study is compared with other work performed in this area. Conclusions and directions for future research are presented in Section 7.

2. Background

In this section we discuss the metrics collection process and methods used in our statistical analysis. We also introduce Alberg diagrams, which are employed for evaluating the resulting models.

2.1. Object-oriented metrics and models

As mentioned, we use the relatively simple and well-understood CK metrics suite. This set of six metrics shows a good potential as a complete measurement framework in an object-oriented environment [36]. Depth of inheritance tree (DIT) for a class corresponds to the maximum length from the root of the inheritance hierarchy to the node of the observed class. Another metrics related

to inheritance is the number of children (NOC), representing the number of immediate descendants of the class in the inheritance tree. Coupling between objects (CBO) is defined as the number of other classes to which a class is coupled through method invocation or use of instance variables. Response for a class (RFC) is the cardinality of the set of all internal methods and external methods directly invoked by the internal methods. We use number of methods (NOM) as a simplified version of more general weighted methods count (WMC), as usually done [3]. The number of internal methods is extracted instead of forming a weighted sum of methods based on complexity. The lack of cohesion in methods (LCOM) is defined as the number of pairs of non-cohesive methods minus the count of cohesive method pairs, based on common instance variables used by the methods in a class.

In addition to CK metrics, we also collect the number of source lines of code (LOC), since we want to verify how well early life cycle models compare with late lifecycle models, which are usually based on size or a proxy of it [13]. The CK metrics can be collected during analysis and design, while LOC is available only after development.

Since the analyzed code is written in C++, LOC is computed counting the number of semicolons.

The issue of object-oriented software metrics has been discussed in details in books by Henderson-Sellers [23] and Whitmire [47]. Various object-oriented metrics have been proposed. Some of them account for deficiencies of the CK suite [30]. The metric suite proposed by Li [30] consists of the number of ancestor classes (NAC), number of local methods (NLM), class method complexity (CMC), number of descendent classes (NDC), coupling through abstract data type (CTA), and coupling through message passing (CTM). Marchesi [34] introduces metrics for object oriented analysis models in UML. Nesi and Querci [38] propose a set of complexity and size metrics for effort evaluation and prediction, providing also a validation for some of them. Reyes and Carver [41] define an object-oriented inter-application reuse measure. Shih et al. [45] propose a concepts of unit repeated inheritance and inheritance level technique for measuring the software complexity of an inheritance hierarchy. Bansiya and Davis [2] introduce average method complexity (AMC) and class design entropy (CDE) that measure the complexity of a class using the information content. Kamiya et al. [26] propose a revised set of CK metrics for software with reused components. Miller et al. [37] propose four new measures of inheritance, identity, polymorphism, and encapsulation in object-oriented design. Teologlou [46] describes the predictive object points for size and effort estimation.

The set of object-oriented design metrics and the source lines of code count used in this paper are collected from the source code using WebMetrics, a software metrics collection system [44].

There has been a lot of work in this area, both theoretical and empirical [5]. Li and Henry [31] employed the CK suite without CBO for predicting maintainability of two commercial software systems developed in Ada. Basili et al. [3] and Briand et al. [6] evaluate the ability of CK suite to predict fault-prone classes. Chidamber et al. [10] explored the impact that CK metrics and program size have on the development time for three financial applications. Briand and Wüst [7] investigated the influence the CK metrics have on the time spent in development. They used data from a music editor developed in the university. Ronchetti and Succi [42] used the subset of CK metrics, extracted from the analysis documentation of two telecommunication projects, to model size of the system. An overview of the related work in this area with comparison of the most important aspects of the different studies is shown in Table 6.

Our paper contributes to the existing research on this subject by (a) focusing on a specific application domain (b) using novel statistical methods applied to (c) data from industrial projects.

2.2. Issues in modeling object-oriented systems

In this study, we use as dependent variable the number of defects for a class, a count variable measured on an absolute scale.

Briand et al. [4] claim that some theoretical prescriptions on the suitability of parametric statistical analysis on this kind of data represent a substantial obstacle in empirical software engineering and are not really required to obtain useful, usable models.

They base their arguments on earlier studies by statisticians. Mayer [35] demonstrates that treating ordinal data as interval can be inappropriate, leading to underestimation of relationships between the variables. In response to this, Labovitz [28] states that Mayer's findings are applicable only to dichotomized scales, which cluster and almost overlap data on one end, and stretch the data in the other parts of the scale.

Although there is no doubt that some analyses could provide useful results even though all theoretical assumptions are not met, it is very important to apply those statistical methods with the closest assumptions to the empirical system. This is especially critical having in mind that inappropriate methods, such as the common practice of treating count variables as continuous, may result in inefficient and biased models and wrong or misleading results [15,21]. Discreteness of the dependent variable leads to conservative confidence intervals, resulting in overestimated significance level for dependent variables [33].

In this paper suitable statistical models are investigated in detail, such as the negative binomial model and its zero inflated extension.

2.2.1. Regression models for the count data

The popular regression models require specific assumptions about the distribution of the analyzed data [18]. Linear regression model, for example, requires normal distribution of the error and homoscedasticity. With software engineering data, most of the necessary assumptions are usually not fully satisfied. For such cases, alternative regression procedures appear more appropriate.

The most common distributions for count data are Poisson and multinomial distributions [27,32]. The Poisson distribution is particularly suitable for counting events occurring over time. In the corresponding PRM, the Poisson distribution determines the probability of a count, where the mean of the distribution is a function of the independent variables. PRMs have been used in software engineering for modeling number of faults [20] and effort [7].

PRM requires equidispersion, i.e., equality of the conditional variance and the conditional mean of the dependent variable. When conditions for the PRM are not met, e.g., in case of high conditional variance of the dependent variable, i.e., higher probability of low and high counts, the negative binomial (NB) distribution and the associated NBRM can be used [32,7].

It is common in software metrics data that the number of zeros exceeds the prediction of both PRM and NBRM. Zero-inflated count models that explicitly model the number of predicted zeros have been proposed in other disciplines, such as manufacturing [29], but have not been applied in software engineering yet.

In the following three sections we provide more details about PRM, NBRM, and ZINBRM.

2.2.2. Poisson regression model

A Poisson process is a model describing the occurrence of events over time [40]. It assumes that an occurrence of an event in an interval depends only on the size of the interval and not on the history of the process. A Poisson distribution is the distribution of the numbers of events resulting from a Poisson process.

The Poisson distribution for a dependent variable y, and a vector of n independent variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ is given by

$$\Pr(y|\boldsymbol{x}) = \frac{e^{-\mu} \cdot \mu^y}{y!}$$

where μ is the conditional mean value of y, $\mu = E(y|\boldsymbol{x}) = \mu(y|\boldsymbol{x})$.

The Poisson distribution requires equidispersion of data, that is, the conditional mean and the conditional variance of the dependent variable should be equal [7,32]: $\mu(y|\boldsymbol{x}) = \mathrm{Var}(y|\boldsymbol{x})$.

In practice, the conditional variance of the dependent variable in the model is often higher than its conditional mean, i.e., the dependent variable is *overdispersed*. The main cause of overdispersion is the failure of the Poisson distribu-

tion to account for heterogeneity in the data. Overdispersion seriously compromises the goodness of fit of the model [32], resulting also in over-estimated statistical significance of the predictors [8].

The PRM accounts for heterogeneity in the data based on the observed characteristics of items, i.e., based on the independent variables. The goal of statistical analysis is to find simple regression functions that accurately model the behavior of the data.

The exponential regression function is commonly used in PRM [33,32]. It corresponds to the multiplicative model for the means. The conditional mean is given by

$$\mu(y|\boldsymbol{x}) = \mathrm{e}^{\beta_0+\beta_1 x_1+\cdots+\beta_n x_n} = \mathrm{e}^{\boldsymbol{x}\boldsymbol{\beta}}$$

where $\boldsymbol{\beta}$ is the vector of model parameters.

2.2.3. Negative binomial model

Empirical data in software engineering are often overdispersed, sometimes with excess of zeros. Main sources of poor fit are models' inability to adapt to small counts or large counts, and lack of complete control over experiments. A great part of the variability in the observed data is attributed to unknown sources—a problem usually referred to as "unexplained heterogeneity." In addition to this, excessive occurrence of zero counts also represents an obstacle in building models.

An extension of the PRM, the negative binomial regression model, allows the conditional variance of the dependent variable to exceed the conditional mean. The negative binomial model has been proposed to deal with overdispersed data in software engineering [7].

The NBRM can be derived from the Poisson distribution taking into account the unobserved heterogeneity [19]. It models the effect of unobserved variables omitted from the original model. In the NBRM, the mean μ is replaced with the random variable $\tilde{\mu}$:

$$\tilde{\mu} = \mathrm{e}^{\boldsymbol{x}\boldsymbol{\beta}+\varepsilon}$$

where ε is a random error uncorrelated with $\boldsymbol{x}$. ε represents the unobserved heterogeneity.

The relationship between $\tilde{\mu}$ and the original μ is

$$\tilde{\mu} = \mathrm{e}^{\boldsymbol{x}\boldsymbol{\beta}}\mathrm{e}^{\varepsilon} = \mu\mathrm{e}^{\varepsilon}$$

With the assumption that $E(\varepsilon) = 0$, the expected count after adding the new source of variation is the same as it was for the PRM, i.e., $E(\tilde{\mu}) = \mu$.

For simplicity we rewrite this formula as $\tilde{\mu} = \mu\delta$, where $\delta = \mathrm{e}^{\varepsilon}$.

In a NBRM, for a given combination of independent variables, there is a distribution of $\tilde{\mu}$'s rather than a single value. Consequently, the probability distribution function for $\delta = \mathrm{e}^{\varepsilon}$ must be specified to determine the probability for

the dependent variable. The resulting distribution is a combination of the Poisson distribution and the probability distribution for δ.

$$\Pr(y|\boldsymbol{x}) = \int_0^\infty [\Pr(y|\boldsymbol{x},\delta)\cdot\Pr(\delta)]\mathrm{d}\delta$$

Due to the closed form of the result [33] and the particular suitability to represent Poisson processes, the gamma distribution with positive parameter v, g_v, is commonly used for δ:

$$g_v(\delta) = \frac{v^v}{\Gamma(v)}\delta^{v-1}\mathrm{e}^{-\delta v} \quad \text{for } v > 0$$

where Γ is the Euler gamma function:

$$\Gamma(x) = \int_0^\infty t^{x-1}\cdot\mathrm{e}^{-t}\mathrm{d}t$$

The resulting combined NB distribution is given by [33]:

$$\Pr(y_i|\boldsymbol{x}_i) = \frac{\Gamma(y_i+v)}{y!\cdot\Gamma(v)}\cdot\left(\frac{v}{v+\mu_i}\right)^{\mu}\cdot\left(\frac{\mu}{v+\mu_i}\right)^{y_i}$$

For the NB distribution, the conditional mean of the dependent variable is the same as for the PRM while the conditional variance of the dependent variable is quadratic in the mean μ:

$$\mathrm{Var}(y|\boldsymbol{x}) = \mu\left(1+\frac{\mu}{v}\right)$$

Since v is positive [33] and μ for count variables is also positive, the variance exceeds the conditional mean of the original Poisson distribution.

The term $\alpha = \frac{1}{v}$ is usually referred to as the "dispersion parameter," since increasing α increases the conditional variance of y. Consequently, a low value of α represents a low level of over-dispersion.

With increased α, the probability of zero values in the NB is increased. For sufficiently large α, the conditional mode for all the values of the independent variables becomes equal to 0.

The NB distribution corrects three main sources of poor fit that are often found when the Poisson distribution is used (Fig. 1).

First, the variance of the NB-distributed dependent variable exceeds the corresponding variance of the Poisson distribution for the given mean.

Second, the increased variance in the NB results in substantially larger probabilities of small counts.

Third, the probability for larger counts is slightly larger.

In the first part of the graph, the probability of low values given by the NB distribution is higher than the one given by the Poisson distribution. A similar behavior of the two distribution functions can be observed for the larger

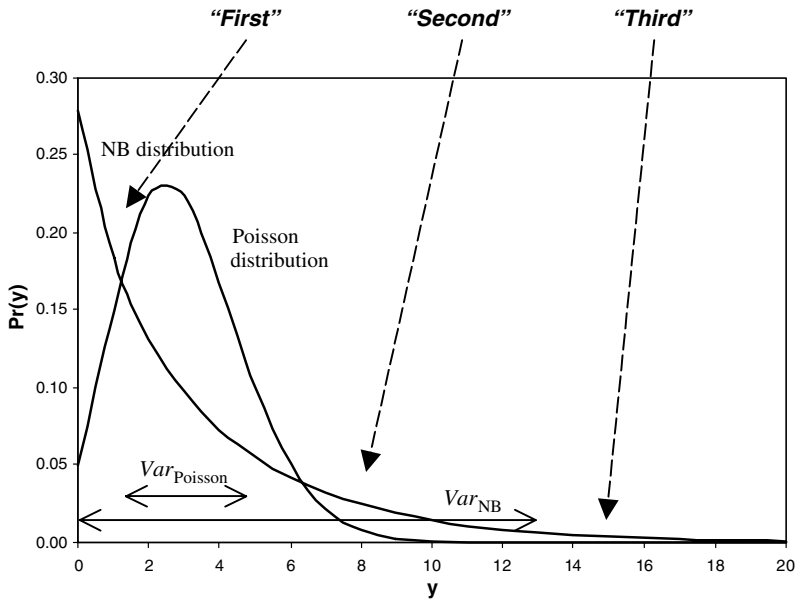


Fig. 1. Poisson and NB distribution for $\mu = 3$.

counts, where NB distribution also has higher values. For a given mean $\mu = 3$ in both distributions and for $v = 0.84$ in NB distribution, $\mathrm{Var}_{\mathrm{NB}} = 13.69$ is greater than $\mathrm{Var}_{\mathrm{Poisson}} = 3$.

Still, the NBRM has only limited success in prediction of large zero counts. The zero-inflated models are more appropriate for such cases.

2.2.4. Zero-inflated model

Zero-inflated models take into explicit account the occurrences of zero counts [33]. These models also increase the conditional variance of the dependent variable. While they have been used successfully in other fields, such as manufacturing [29], up to the knowledge of the writers they have not been applied to software defects data.

Zero-inflated models are based on the idea that there are two different processes generating the population: one producing zero counts and one producing positive counts (Fig. 2). The overall population is partitioned in two groups: a group of items with strictly zero counts and the other group with non-zero probability of counts different than zero. An item belongs to the first group with probability ψ and to the latter with probability $1 - \psi$. This probability is determined by the independent variables in the model [29,22]. The concept of two groups represents a discrete, unobserved heterogeneity since it is not known to which of the two groups an item with a zero count belongs.

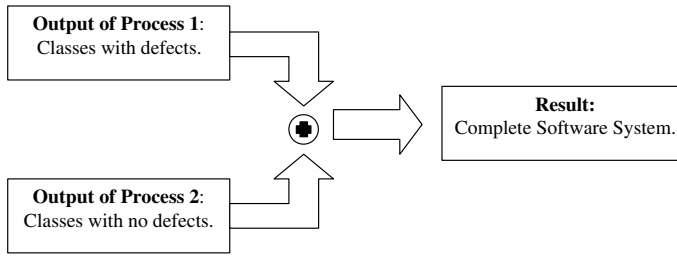


Fig. 2. Zero inflated model of software development.

In the group of items that are not always equal to zero, the resulting counts are governed by the PRM or the NBRM. For the NBRM, zeroes in this group occur with probability:

$$\Pr(y = 0|\boldsymbol{x}) = \left(\frac{v}{v+\mu}\right)^{\mu}$$

where $\mu_i = e^{\boldsymbol{x}i\boldsymbol{\beta}}$.

The overall probability of zeroes is the sum of the probabilities of zeroes in each group multiplied by the probability of an item belonging to a particular group.

The resulting model has the following form:

$$\begin{cases} \Pr(y = 0|x) = \psi(x) + (1 - \psi(x))P_M(y = 0|x) \\ \Pr(y \neq 0|x) = (1 - \psi(x))P_M(y \neq 0|x) \end{cases}$$

where $P_M(y|\boldsymbol{x})$ is the probability given by the PRM or NBRM. In this study, we use zero-inflated model based on NBRM.

In the combined zero-inflated model, the probability ψ is determined by either a probit or a logit model $\psi = F(\boldsymbol{z}\boldsymbol{\gamma})$, where F is the normal or the logistic cumulative distribution function, respectively. Because of the closed form of the result, we follow the common practice of choosing the ZINBRM based on the logit model [33]:

$$\psi(z) = \frac{e^{\gamma \cdot z}}{1 + e^{\gamma \cdot z}}$$

The predictor vector $\boldsymbol{z}$ can (but does not have to) be the same as the vector $\boldsymbol{x}$ in the original PRM or NBRM. Clearly, if $\boldsymbol{z}$ and $\boldsymbol{x}$ are equal, i.e., if the same predictors are used in both parts of the model, the resulting model has twice as many parameters as the corresponding NBRM.

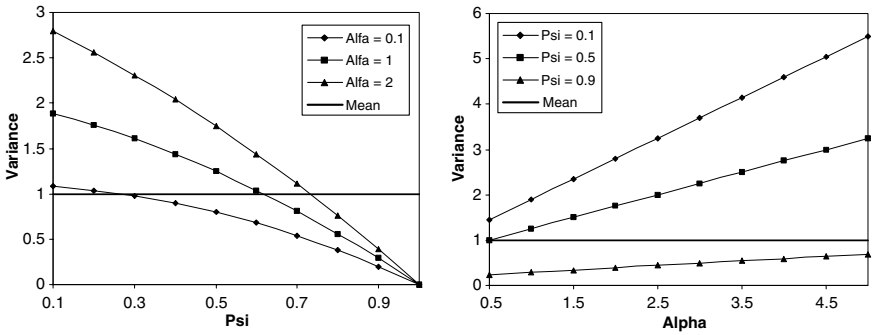


Fig. 3. Variance of the ZINBRM as a function of ψ and α.

The variance of the ZINBRM is given by [22]:

$$\mathrm{Var}(y|\boldsymbol{x},\boldsymbol{z}) = E[\mathrm{Var}(y|\boldsymbol{x},\boldsymbol{z})] + \mathrm{Var}[E(y|\boldsymbol{x},\boldsymbol{z})] = \mu(1-\psi)[1+\mu(\psi+\alpha)]$$

The variance of the ZINBRM typically exceeds variance of the PRM for non-zero ψ. A comparison of variance of the ZINBRM with PRM for a fixed mean $\mu = 1$ and different values of ψ and α is presented in Fig. 3.

2.3. Evaluation of the goodness of fit of a model

To evaluate the goodness of fit of a model, several criteria are often used.

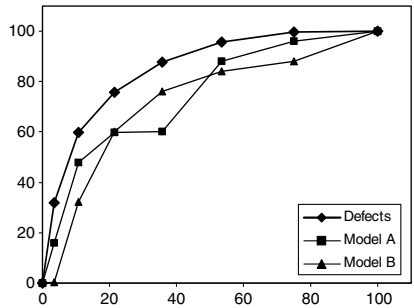
First, a model must satisfy the hypotheses under which it has been developed. Standard OLS linear regression models require a check of linearity, homoscedasticity, constant variance, and gaussian noise. In addition to the "standard" checks, PRM, NBR, and ZINBR require the dispersion of the data to be captured properly, that is, absence of overdispersion.

Second, a model has to be as close as possible to the actual data. Correlation coefficients or mean square errors are used for such purpose.

Third, a model has to satisfy the goal for which it has been developed. Graphical methods, such as Alberg diagrams, can be used to determine the effectiveness of a model. In our case, Alberg diagrams can be used to compare performance of the different models in terms of identification of classes with defects [39].

Alberg diagrams assume a Pareto distribution of defects [11], i.e., a small number of classes contains the majority of defects and consequently consumes most of the effort in debugging the system. An Alberg diagram is built ordering classes in decreasing number of defects. The x-axis is the percentage of the total number of classes inspected so far, and the y-axis is the percentage of the total number of defects discovered in the corresponding classes.

By comparing the curves formed using the *observed data* and the results *estimated* by the model, the effectiveness of the model in identifying critical classes



Defects		Model A		Model B	
C1	80	*C3*	40	*C7*	1
C2	70	*C1*	80	*C1*	80
C3	40	*C4*	30	*C2*	70
C4	30	*C7*	1	*C3*	40
C5	20	*C2*	70	*C5*	20
C6	10	*C5*	20	*C6*	10
C7	1	*C6*	10	*C4*	30

Fig. 4. Example of Alberg diagram.

can be visually assessed. It is reported that models with lower correlation coefficients can provide better prediction with respect to the criticality of the analyzed classes [39].

A simple application of Alberg diagrams is presented in Fig. 4. There are seven classes (C1–C7) with the corresponding defect counts. We have two different models—A and B. Model A has a higher correlation with the empirical data (0.5) than model B (0.2). However, in the range from 20% to approximately 60% of the defects, model B performs better if we use the model for predicting the most defect-prone classes.

3. Discussion of the experimental data

This research focuses on five projects developed by a North-American company that prefers to remain anonymous. The development and testing spanned over approximately five years. The projects are developed for embedded systems in a real-time telecommunication domain, mainly using the C++ programming language.

The development teams of the projects had equivalent skill-sets. The developers assigned to each project had similar experience and education levels equivalent to a B.Sc. in Electrical and Computer Engineering or in Computer Science. No particular pattern was observed in the assignment of developers to products, suggesting no selection bias.

Total size of the five projects is 63,394 lines of code distributed in 395 classes (Table 1).

The CK metrics and LOC are collected from the source code using WebMetrics, a software metrics collection tool [44].

A summary of the descriptive statistics for the extracted metrics is provided in Table 2. We find that both metrics dealing with inheritance—depth of inheritance tree and number of children, assume low values. Chidamber et al. [10] and Ronchetti and Succi [42] observed similar behaviors.

Table 1
Number of available data points for projects

Project	Number of classes	LOC
A	93	8802
B	119	6496
C	101	24,989
D	44	5316
E	38	17,791

Table 2
Descriptive statistics of the extracted metrics and fixes for the projects

	Project A				Project B			
	Min	Max	Mean	Std. Dev.	Min	Max	Mean	Std. Dev.
LOC	2	754	94.65	129.11	1	1128	57.24	156.86
NOM	0	54	9.78	9.29	0	74	9.24	13.79
DIT	0	4	0.90	1.27	0	3	0.97	1.12
NOC	0	11	0.27	1.39	0	5	0.16	0.62
CBO	0	68	11.68	12.17	0	63	4.17	8.02
RFC	0	122	24.59	25.93	0	246	17.59	35.36
LCOM	0	821	59.44	115.94	0	1540	87.90	234.98
Defects	1	28	5.28	4.71	0	22	2.35	5.28
	Project C				Project D			
LOC	1	1377	247.42	302.98	2	1674	120.82	258.24
NOM	1	219	32.60	41.13	0	94	16.09	19.26
DIT	0	2	0.97	0.96	0	1	0.25	0.44
NOC	0	11	0.16	1.20	0	4	0.14	0.63
CBO	0	121	23.17	24.09	0	111	11.05	18.24
RFC	2	283	67.46	71.90	0	336	33.23	52.38
LCOM	0	23247	1041.30	3197.59	0	3897	256.57	701.88
Defects	0	24	1.38	3.12	0	16	1.30	2.92
	Project E							
LOC	2	3181	468.18	611.86				
NOM	0	188	41.61	43.41				
DIT	0	2	0.26	0.55				
NOC	0	2	0.05	0.32				
CBO	0	124	17.82	22.67				
RFC	0	418	69.05	75.19				
LCOM	0	17,048	1606.03	3585.45				
Defects	0	32	5.55	7.03				

The number of revisions for a class is the external measure used as the dependent variable in our analysis. The company employs a rigorous process to record such revisions in a defect database and in a configuration management system. As commonly done in software engineering, we assume that

the number of revisions closely corresponds to the number of defects found in a class [17]. In this paper, we will use these two terms interchangeably.

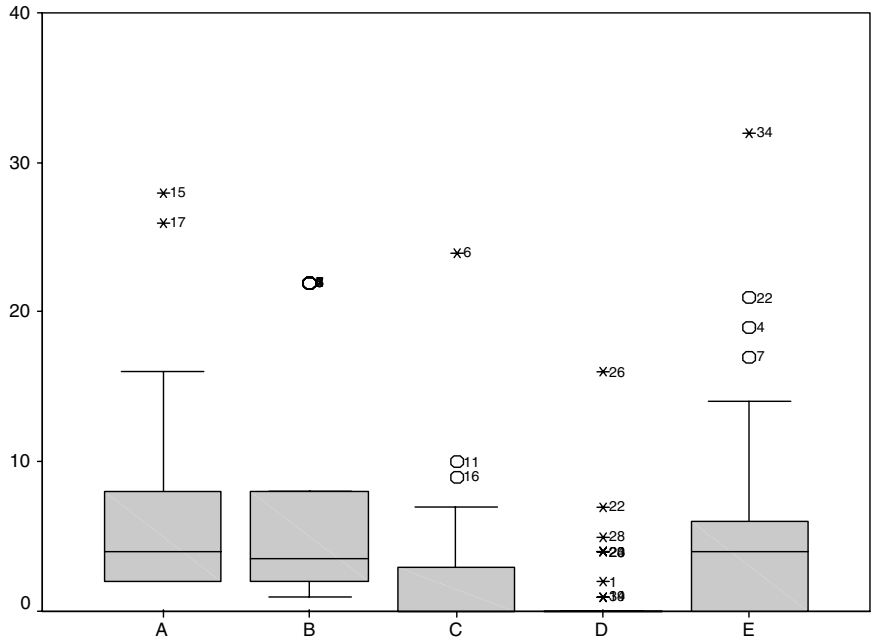


Fig. 5. Boxplots for distribution of the number of defects.

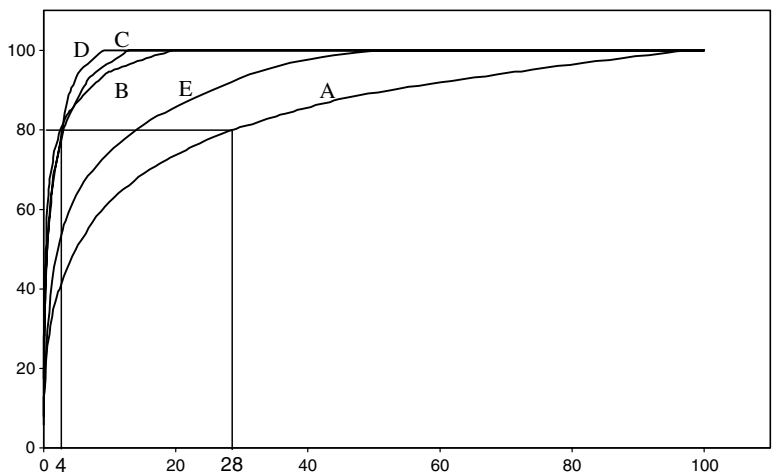


Fig. 6. Cumulative number of defects for the five projects.

The number of defects found in a class ranges from 0 to 32, with most of the classes having very low number of defects. The boxplots of the observed data evidence the non-normal, Poisson-like, nature of the distributions of defects (Fig. 5). The number of defect counts for project D is very low resulting in the boxplot for this project to concentrate around zero.

As observed by Ebert and Baisch [11], the distribution of defects often follows the Pareto rule: a relatively small portion of the system classes causes most of the defects. As mentioned, Alberg diagrams evidence this phenomenon. Fig. 6 shows the observed number of defects for the five projects in the form of Alberg diagrams. It 80% of the defects are caused by only 3% of the classes in projects B and F, 4% in project C, 14% in project E, and 28% in project A.

4. Extraction of the models

To determine the most suitable models, we proceed as follows.

First, the best predictors are identified using the Spearman rank correlation between the number of defects and the internal metrics (Table 3).

Often relations between internal metrics and quality can be explained in terms of size metrics [13]. In this study, we use RFC and NOM, which are size metrics in the sense of [3]. We also use LOC to determine the information that is lost using the early available design-based models instead of later code-based models.

DIT and NOM are not taken into further consideration due to the low correlation coefficients and relatively narrow range of values, which poses strong constraints on the statistical techniques to use.

Then, we proceed with the more advanced statistical models to deal with non-normal distribution of the dependent variable and other specifics of the data. We apply PRM, NBRM, and ZINBRM. Due to the potentially high cross correlation among some of the predictors [9], we use only univariate models [1].

Table 3
Correlations between number of defects and internal metrics for the projects

	CBO	DIT	LCOM	NOC	NOM	RFC	LOC
A	0.00	−0.07	0.11	−0.20	0.14	0.11	0.08
B	0.36*	0.59*	0.24	−0.13	0.22	0.31	0.24
C	0.43*	0.24	0.46	0.03	0.41*	0.43*	0.46*
D	0.19	0.05	0.16	−0.17	0.18	0.25	0.24
E	0.45	0.06	0.32	−0.21	0.31	0.45	0.52

Here and in the following tables a "*" indicates a significance at the .05 level.

The parameters of the PRM are estimated using both the method of ordinal least squares (OLS) and the method of maximum likelihood (ML). The application of the OLS method for fitting the parameters of the generalized linear regression model is justified if the error distribution is normal. In this case, the OLS estimation and the ML estimation of model parameters $\boldsymbol{\beta}$ are approximately the same. The ML method provides a very general solution for fitting of the model parameters with less stringent requirements on the noise. The ML estimator is consistent, i.e., the probability that the ML estimator differs from the true parameter by an arbitrary small amount tends toward zero as the sample size grows. The variance of the ML estimator is the smallest possible among consistent estimators. This feature is usually referred to as *asymptotic efficiency* of the ML estimator.

The likelihood function of the ML estimation for PRMs is

$$l(\boldsymbol{\beta}) = \sum_{i=1}^{k} (y_i \cdot \ln(\mu_i(\boldsymbol{\beta})) - \mu_i(\boldsymbol{\beta}))$$

Table 4
Performance of the models with respect to correlation and dispersion

		RFC		NOM		LCOM		CBO		LOC	
		r	α	R	α	R	α	r	α	r	α
A	PRM OLS	0.51	17.31	0.43	0.33	0.50	2.75	0.10	8.29	0.08	11.39
	PRM ML	0.35	0.32	0.40	0.33	0.49	0.30	0.18	0.36	0.17	0.35
	NBRM	0.34	0.32	0.28	0.11	0.47	0.30	0.18	0.34	0.17	0.35
	ZINBRM	0.34	0.32	0.28	0.02	0.48	0.23	0.18	0.34	0.17	0.17
B	PRM OLS	0.39*	0.96	0.18	2.11	0.01	9.80	0.21*	0.27	0.02	33.30
	PRM ML	0.39*	0.45	0.15	0.26	0.07	0.25	0.19*	0.17	0.11	0.83
	NBRM	0.39*	0.32	0.11	0.07	0.08	0.84	0.11	0.83	0.13	0.05
	ZINBRM	0.30*	0.29	0.12	0.05	0.07	4.24	0.07	4.01	0.12	0.05
C	PRM OLS	0.55*	4.87	0.21*	3.45	0.29*	2.86	0.73*	2.98	0.80*	27.84
	PRM ML	0.55*	4.18	0.19*	2.96	0.27*	2.49	0.66*	0.46	0.83*	2.96
	NBRM	0.55*	3.33	0.26*	0.31	0.82*	0.60	0.68*	0.39	0.83*	2.52
	ZINBRM	0.55*	2.91	0.15*	0.16	0.82*	0.46	0.67*	0.31	0.83*	1.34
D	PRM OLS	0.16	11.02	0.22	5.55	0.17	7.22	0.17	4.73	0.15	5.82
	PRM ML	0.19	7.50	0.22	5.55	0.20	0.69	0.19	0.57	0.16	5.64
	NBRM	0.18	5.62	0.22	0.54	0.19	0.60	0.19	0.76	0.16	3.93
	ZINBRM	0.17	5.44	0.23	0.48	0.20	0.43	0.19	0.76	0.16	0.62
E	PRM OLS	0.65	1.28	0.26	1.46	0.40	1.37	0.73	1.13	0.72*	1.10
	PRM ML	0.65	1.19	0.25	1.36	0.39	1.28	0.72	1.05	0.72*	1.03
	NBRM	0.65	0.50	0.23	0.10	0.39	0.12	0.72	0.79	0.71*	0.77
	ZINBRM	0.65	0.50	0.23	0.03	0.39	0.04	0.72	0.59	0.71*	0.57

We use only the ML method for the estimation of the NBRM and ZINBRM. In the ZINBRM, the logit model is used for prediction of zeros in combination with the NBRM.

The likelihood equation for the ML estimator for the NBRMs is

$$L(\boldsymbol{\beta}|\boldsymbol{y},\boldsymbol{x}) = \prod_{i=1}^{N} \Pr(y_i|\boldsymbol{x}_i)$$

The corresponding maximum likelihood method is also available for zero-inflated models [33].

The vector of model parameters $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ is determined maximizing the cumulative probability over the available data points.

Table 4 presents a summary of the correlation coefficient r and dispersion parameter α for the resulting models.

5. Analysis of the results

The performances of the models reported in Table 4, differ significantly.

In general, ML models exhibit reduced overdispersion, sometimes at the expenses of a reduced correlation coefficient. Within the ML models, the ZINBRM performs the best overall.

As mentioned, the ZINBRM assumes that the overall population is formed by two independent groups, one with zero defects and one with a negative binomial defects distribution. The predominance of the ZINBRM provides the statistical confirmation of two facts already observed in the empirical software engineering community:

(a) Defects tend to concentrate in a few classes, leaving unaffected a (sometimes large) portion of the software systems [12,43].
(b) Defects tend to be distributed according to a negative binomial curve [7].

Staying within univariate models, the best performances overall is provided by RFC and CBO, which sometimes overperform even LOC, a metrics collected only after development.

RFC is simple to define, unambiguous to count, and it has an intuitive meaning easy to translate into statements and prescriptions for developers and testers. Therefore, we have selected it as the basis for our further analysis.

RFC is a measure with a relatively wide range of values (Fig. 7). In our case RFC ranges from 0 to 418. Such a wide range suggests that RFC could be successfully used as a predictor in building regression models. As mentioned, RFC measures the communication between a class and other classes. It counts the

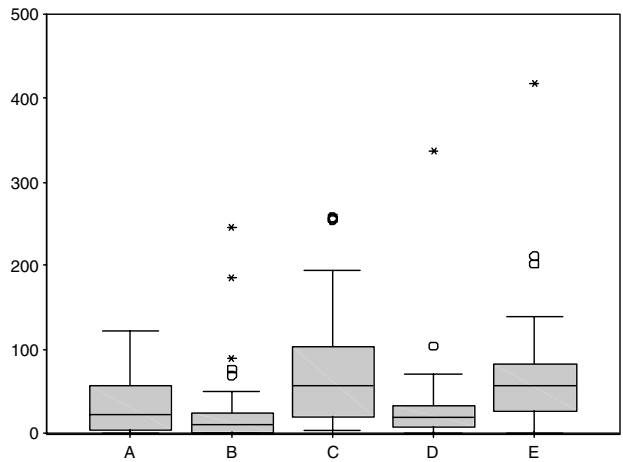


Fig. 7. RFC boxplots for the five projects.

methods of a class and the methods invoked by methods of a class. The complexity of the class increases with its methods and the outside methods invoked.

Clearly, the higher such communication for a class, the higher will be the probability of introducing defects and, consequently, the higher the need for modifications in that class. Moreover, testing and fixing of such class requires more effort by both testers and developers.

The presence of a low number of classes with high RFC values also supports the fact that small number of classes accounts for most of the defects in the system. Managing this aspect of object-oriented design is important for successful allocation of testing and fixing effort. To further investigate this issue, we use Alberg diagrams.

As mentioned, Alberg diagrams are used to compare graphically the performances of the different models with respect to the criticality of the prediction [39]. The Alberg diagrams for the five projects using the four models based on RFC are presented in Fig. 8. The performances of the models for project E are practically the same: their curves are almost overlapped and indistinguishable.

The percentages of the classes to inspect according to RFC, LOC, and an ideal model are presented in Table 5. ZINBRM are used for RFC and LOC. The ideal model is an a posteriori analysis of the lowest percentages of classes needed to generate 80% of the defects; it is our criteria for optimality.

RFC-based models have performances only slightly lower than LOC-based, confirming the effectiveness of these kinds of early lifecycle models. The comparison of the performances of RFC- and LOC-based models with those of the ideal model shows that there is still ample room for improvement. Future

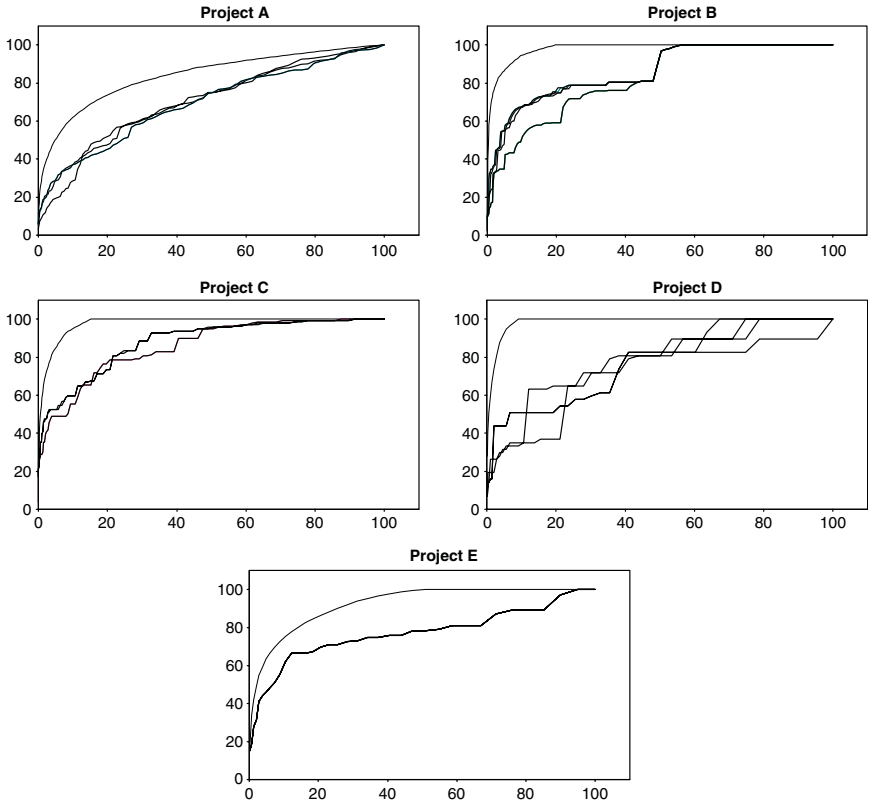


Fig. 8. Alberg diagrams for the RFC-based models.

Table 5
Percentage of classes to inspect for detecting 80% of the defects

	Project A (%)	Project B (%)	Project C (%)	Project D (%)	Project E (%)
Ideal	28	3	4	3	14
RFC	58	44	22	41	59
LOC	58	39	18	35	40

research will evidence whether such improvements are achievable at all and whether using early lifecycle metrics only.

6. Relationships with other studies

An overview of some of the most significant works in this area is presented in Table 6.

Table 6
Related studies

Study	Li and Henry [31]	Basili et al. [3]	Chidamber et al. [10]	Briand and Wüst [7]	Ronchetti and Succi [42]	This work
Environment	Industrial	University	Industrial	University	Industrial CMM level 3	Industrial
Lifecycle phase of the independent variable	Design & code	Code	Design & code	Design & code	Analysis	Design & code
Programming language	Classic-ADA	C++	C++	C++	C++	C++
Application domain	User interface and scientific	Information system	Financial application	Music editor	Telecom	Real-time telecom
Size	2 projects	8 projects	3 projects	1 project	2 projects	5 projects
Dependent variables	Number of lines changed during maintenance as a proxy for maintenance effort	Fault probability	Productivity, rework effort, design effort	Development time as a proxy for effort	Size as proxy for development effort	Number of defects for a class
Statistical analysis	Parametric linear regression	Logistic regression	Stepwise linear regression including dummy variables	Poisson regression and regression trees	Parametric and non-parametric correlation: linear regression	Poisson, NB, and ZINBRM regression
Conclusions	No other CK metrics but CBO influence significantly the dependent variable	NOM, DIT, CBO, RFC significantly influence the dependent variable	High values of CBO and LCOM significantly influence the dependent variable	Size measures can be used as good predictors of effort	NOM significantly influences the dependent variable	Overall RFC has dominant statistically significant effect to the dependent variable
Other considerations	N/A	The cross-correlation of the CK metrics are low	DIT and NOC assume low values. CBO, NOM, and RFC are highly correlated; the other correlations between metrics are low	Coupling measures do not substantially improve quality of the models	DIT and NOC assume low values; the cross-correlation between the three CK metrics are low	LCOM, CBO, and DIT also have significant influence to some of the analyzed projects

As these investigations use different dependent variables and different explanatory metrics, it is not possible to perform quantitative comparisons of the results or to infer solid generalizations. Still, all the dependent variables used in these papers try to explain various aspects of software development effort and quality using design measures.

Although the dependent variables are often heavily skewed and non-normally distributed [31,10,42], only Briand and Wüst take this aspect into account and apply Poisson regression. For the same reasons, Basili et al. [3] decide not to use defect count or density but a binary dependent variable representing the probability of defects for a class. Li and Henry [31] do not clearly specify the form of their model.

The metrics used in the papers are not uniform. Ronchetti and Succi [42] use only subset of CK suite available to them, while Briand and Wüst [7] use variety of other metrics in combination with the CK suite.

Basili et al. [3] reports a considerable influence of RFC on the dependent variable. Chidamber et al. [10] find CBO and LCOM the dominant factors in explaining dependent variables. Li and Henry [31] identify a significant influence of CBO on maintenance effort.

This work uses Alberg diagrams; these diagrams have already been applied in software engineering—see for instance [24], but not for assessing the effectiveness of object oriented models.

This research is the only one focused on the real-time, telecommunication domain. The paper by Ronchetti and Succi [42] uses data from the telecommunication domain, but it does not deal with real-time applications; it reports that NOM appears the dominant factor in explaining software size as a proxy of development effort.

This work is the first building zero-inflated statistical models for software engineering data. These models explain the well-known fact that only a fraction of the classes are responsible for most of the defects in software systems [43].

7. Conclusion

In this paper we evidence that early lifecycle metrics can be used for identifying the most defect prone classes in the context of real-time, telecommunication software systems developed using C++.

To this end, we adopt statistical models applicable to count data, i.e., PRM, NBRM, and ZINBRM. These models account for the typical problems with the software metrics data, such as overdispersion and heterogeneity. The CK metrics and LOC are used as independent variables. Number of modifications as a proxy for defects is used as the dependent variable. The performances of the models are evaluated using correlation coefficients, overdispersion parameters, and Alberg diagrams.

Altogether, in the analysed datasets zero-inflated negative binomial regression models based on RFC provide the best descriptive ability, similar to that of lines of code –a late lifecycle size metrics.

These results confirm and formally model early, mainly anecdotal evidence claiming that:

(a) Communication between classes increases the probability of defect for such classes.
(b) There are several classes without any defect.
(c) The defects are distributed according to a negative binomial distribution.

These findings can help project managers in identifying the classes and files that require more careful design, refactoring, and rigorous testing.

Evidently, more research is necessary in this field. *First*, it would be interesting to replicate this experiment to see if it is possible to obtain the same results. The comparison could be conducted using suitable statistical techniques, such as meta-analysis. *Second*, it would be important to determine the defect-free classes not with a stochastic process but through a deterministic approach. *Third*, multivariate models on larger datasets could be attempted, seeking for more precise predictions. *Finally*, the results would be even more significant if we were able to collect them from *actual* number of defects or defect removal effort.

Acknowledgements

The authors acknowledge the support of the Natural Science and Engineering Research Council of Canada, the Government of Alberta, the University of Alberta, and the Free University of Bozen. Special thanks also go to Eric Liu for his contribution to this work. Thanks also to Luigi Benedicenti, Snezana Djokic, Arrigo L. Frisiani and Forrest Shull for their valuable feedbacks on, and inputs to this work.

References

[1] A. Aron, E.N. Aron, Statistics for the Behavioral and Social Sciences, Prentice Hall, 1997.
[2] J. Bansiya, C. Davis, Design and code complexity metrics for OO classes, Journal of Object Oriented Programming 12 (1) (1999) 35–40.
[3] V.R. Basili, L.C. Briand, W.L. Melo, A validation of object-oriented design metrics as quality indicators, IEEE Transactions on Software Engineering 22 (10) (1996) 751–761.
[4] L.C. Briand, K. El Emam, S. Morasca, On the application of measurement theory in software engineering, Journal of Empirical Software Engineering 1 (1) (1996).
[5] L.C. Briand, S. Morasca, V.R. Basili, Property-based software engineering measurement, IEEE Transactions on Software Engineering 22 (1) (1996) 68–86.

[6] L.C. Briand, J. Daly, V. Porter, J. Wüst, Predicting fault-prone classes with design measures in object-oriented systems, in: 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998.
[7] L.C. Briand, J. Wüst, The impact of design on development cost in object-oriented systems, Technical Report, 1999. Available from: <http://www.iese.fhg.de/network/ISERN/pub/technical_reports/isern-99-16.pdf>.
[8] A.C. Cameron, P.K. Trivedi, Econometric models based on count data: comparisons and applications of some estimators and tests, Journal of Applied Econometrics 1 (1986) 29–93.
[9] S.R. Chidamber, C.F. Kemerer, A metrics suite for object oriented design, IEEE Transactions on Software Engineering 20 (6) (1994) 476–493.
[10] S.R. Chidamber, D.P. Darcy, C.F. Kemerer, Managerial use of object-oriented software: an explanatory analysis, IEEE Transactions on Software Engineering 24 (8) (1998) 629–639.
[11] C. Ebert, E. Baisch, Industrial application of criticality predictions in software development, in: 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 1998.
[12] C. Ebert, Metrics for identifying critical components in software projects, in: Handbook of Software Engineering and Knowledge Engineering, World Scientific Pub. Co., 2001.
[13] K. El Emam, S. Benlarbi, N. Goel, The confounding effect of class size on the validity of object-oriented metrics, Technical Report, NRC/ERB-1062, September 1999.
[14] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison-Wesley, 1999.
[15] N.E. Fenton, M. Neil, A critique of software defect prediction models, IEEE Transactions on Software Engineering 25 (5) (1999) 675–689.
[16] N.E. Fenton, N. Ohlsson, Quantitative analysis of faults and failures in a complex software system, IEEE Transactions on Software Engineering 26 (8) (2000) 797–814.
[17] N.E. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, PWS Publishing Company, 1997.
[18] J.E. Freund, G.A. Simon, Statistics: A First Course, Prentice Hall, 1996.
[19] C. Gourieroux, A. Monfort, A. Trognon, Pseudo-maximum likelihood methods: applications to Poisson models, Econometrica 52 (1984) 701–720.
[20] T. Graves, A.F. Karr, J.S. Marron, H. Siy, Predicting fault incidence using software change history, IEEE Transactions on Software Engineering 26 (7) (2000) 653–661.
[21] A.R. Gray, S.G. MacDonell, A comparison of techniques for developing predictive models of software metrics, Information and Software Technology 39 (1997) 425–437.
[22] W.H. Greene, Accounting for excess zeros and sample selection in Poisson and negative binomial regression models, Working Paper EC-94-10, Stern School of Business, Economics Department, 1994.
[23] B. Henderson-Sellers, Object-Oriented Metrics: Measures of Complexity, Prentice Hall, 1995.
[24] B. Huges, F. Young-Martos, A. Cunliffe, Software fault count and density prediction, in: R. Kusters, A. Cowderoy, F. Heemstra, E. van Veenendaal (Eds.), Project Control for Software Quality, Shaker Publishing, 1999.
[25] W. Humphrey, A Discipline for Software Engineering, SEI Series in Software Engineering, Addison-Wesley, 1995.
[26] T. Kamiya, S. Kusumoto, K. Inoue, Y. Mohri, Empirical evaluation of reuse sensitiveness of complexity metrics, Information and Software Technology 41 (5) (1999) 297–305.
[27] G. King, Statistical models for political science event counts: bias in conventional procedures and evidence for the exponential Poisson regression model, American Journal of Political Science 32 (1988) 838–863.
[28] S. Labovitz, In defense of assigning numbers to ranks, American Sociological Review 36 (1971) 521–522.

[29] D. Lambert, Zero-inflated Poisson regression with an application to defects in manufacturing, Technometrics 34 (1992) 1–14.
[30] W. Li, Another metric suite for object-oriented programming, Journal of Systems and Software 44 (2) (1998) 155–162.
[31] W. Li, S. Henry, Object-oriented metrics that predict maintainability, Journal of Systems and Software 23 (2) (1993) 111–122.
[32] C.J. Lloyd, Statistical Analysis of Categorical Data, Wiley-Interscience, 1999.
[33] J.S. Long, Regression models for categorical and limited dependent variables, Advanced Quantitative Techniques in the Social Sciences, vol. 7, Sage Publications, 1997.
[34] M. Marchesi, OOA metrics for the Unified Modeling Language, in: Second Euromicro Conference on Software Maintenance and Reengineering, Los Alamitos, CA, USA, 1998.
[35] L. Mayer, A note on treating ordinal data as interval data, American Sociological Review 36 (1971) 519–520.
[36] M. Mendonça, V.R. Basili, Validation of an approach for improving existing measurement frameworks, IEEE Transactions of Software Engineering 26 (6) (2000) 484–499.
[37] B.K. Miller, H. Pei, K. Chenho, Object-oriented architecture measures, in: 32nd Annual Hawaii International Conference on Systems Sciences, Los Alamitos, CA, USA, 1999.
[38] P. Nesi, T. Querci, Effort estimation and prediction of object-oriented systems, Journal of Systems and Software 42 (1) (1998) 89–102.
[39] N. Ohlsson, H. Alberg, Predicting fault-prone software modules in telephone switches, IEEE Transactions on Software Engineering 22 (12) (1996).
[40] A. Papoulis, Probability, Random Variables, and Stochastic Processes, McGraw Hill College Div, 1991.
[41] L. Reyes, D. Carver, Predicting object reuse using metrics, in: 10th International Conference on Software Engineering and Knowledge Engineering, Skokie, IL, USA, 1998.
[42] M. Ronchetti, G. Succi, Early estimation of software size in object-oriented environment—a case study in a CMM Level 3 Software Firm, Information Sciences 176 (5) (2006) 475–489.
[43] F. Shull, Summary of the 2nd CeBASE e-Workshop, 2001. Available from: <http://www.cebase.org/www/defectreduction/eworkshop2/eworkshop_2_summary>.
[44] G. Succi, L. Benedicenti, C. Bonamico, T. Vernazza, The Webmetrics Project—Exploiting 'Software tools on demand', in: World Multiconference on Systemics, Cybernetics, and Informatics, Orlando, FL, 1998.
[45] T.K. Shih, Y.-C. Lin, W.C. Pai, C.-C. Wang, An object-oriented design complexity metric based on inheritance relationships, International Journal of Software Engineering and Knowledge Engineering 8 (4) (1998) 541–566.
[46] G. Teologlou, Measuring object-oriented software with predictive object points, in: 10th European Software Control & Metrics Conference, Herstmonceux, England, 1999.
[47] S.A. Whitmire, Object-Oriented Design Measurement, John Wiley & Sons, 1997.