# An Empirical Study of the Impact of Count Models Predictions on Module-Order Models

Taghi M. Khoshgoftaar*
Erik Geleyn
Kehan Gao
Florida Atlantic University
Boca Raton, Florida USA

## Abstract

*Early software quality prediction models are used to achieve high software reliability. Prediction models that estimate a quality factor for software modules can be used in directing corrective efforts early in the life cycle process. Precise quantitative prediction values for the quality factor is often not sufficient. Instead, predicting the rank-order of modules with respect to the quality factor may be more beneficial to the development team. A module-order model (MOM) uses an underlying quantitative prediction model to predict this rank-order.*

*This paper compares performances of module-order models of two different count models which are used as the underlying prediction models. They are the Poisson regression model (PRM) and the zero-inflated Poisson (ZIP) regression model. It is demonstrated that improving a count model for prediction does not ensure a better MOM performance. A case study of a full-scale industrial software system is used to compare performances of module-order models of the two count models. It was observed that improving prediction of the Poisson count model by using zero-inflated Poisson regression did not yield module-order models with better performance. Thus, it was concluded that the degree of prediction accuracy of the underlying model did not influence the results of the subsequent module-order model. Module-order modeling is proven to be a robust and effective method even though both underlying prediction may sometimes lack acceptable prediction accuracy.*

Keywords*: Software reliability, software metrics, module-order modeling, count models, Poisson, ZIP*

## 1. Introduction

The growing importance of software managed systems in today's world is so obvious that every day more human lives and huge economical assets rely on those systems. Because of the possible consequence of a failure, it is crucial to make them as reliable as possible. To achieve those reliability enhancements, software quality prediction models have been developed. A software quality prediction model estimates a quality factor (like number of faults) using the data collected in the early phases of the life cycle. Linear and non-linear regression analysis are some of the simplest methods used as prediction models. Other methods have also been applied to this field, like Case-Based Reasoning [6], Fuzzy logic [14], Genetic Programming [5] or Tree based learning [3].

Project managers are then able to apply the enhancement efforts early in the life cycle to the modules requiring the most attention. These efforts range from extra walkthroughs to extensive reviews. However, most of the time only limited resources are available to perform these tasks and most importantly the same treatment is applied to all the enhanced modules [2]. In such cases there is little interest in knowing the exact number of faults in a given module. The useful information for managers is a ranking starting with the modules with the lowest quality. According to this module ranking, managers can then dispense available resources for quality enhancements until they are exhausted.

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/.

A module-order model answers this problem by predicting the rank-order of modules according to a quality factor. Prediction models that estimate the quality factor are used as underlying models by module-order models to determine the rank-order of modules.

Previous work [2] has proved that module-order models may remain robust even if the underlying model performs poorly as a prediction model. This paper will continue this work by showing the results obtained by using two different count models with a significant difference in their prediction accuracy. Count models are models where the response (dependent) variable values are non-negative integers. They are very common in fields such as economics, social science, and software engineering.

A case study of a full-scale industrial software system compared the performances of the two count models as module-order models. To our knowledge this is the first time such a comparative study is performed for software quality modeling. It was observed that improving prediction of the Poisson count model by using zero-inflated Poisson regression did not yield module-order models with better performance.

The layout of the rest of the paper is as follows. A description of the module-order modeling technique is presented in Section 2. In Section 3, the two different count modeling techniques are presented. Section 4 introduces our methodology while Section 5 and 6, present the results, findings and conclusion of the comparative case study.

## 2. Module-Order Modeling

This section presents module-order modeling starting with the concepts, the ranking results and the evaluation procedure.

### 2.1. Concepts of Module-Order Modeling

While quality factors can only be measured late in the life cycle of a software product, most software metrics may be collected much earlier. These metrics can be used as inputs for a software quality prediction model. The result of the prediction is then used to decide the enhancement process for the candidate modules. The choice of the quality factor is up to the project manager, but should be a good representation of the actual quality of the module. We define a fault-prone module as a module presenting enough faults that makes it eligible for an enhancement treatment.

Module-order models predict the relative quality of each module, especially the most faulty ones. Here we will rank the modules according to the number of faults found during system test. The type of scale normally used ranges from ordinal to absolute. To summarize, a module-order model is composed of the following three components.

1. An underlying software quality prediction model.
2. A ranking of modules according to the quality factor estimated by the underlying prediction model.
3. A procedure for evaluating accuracy of the ranking.

### 2.2. Underlying Software Quality Prediction Model

Every software quality prediction model may be considered as a function of a vector of software measurements, $\mathbf{x}_i$, predicting a quality factor for module $i$, $F_i$, $F_i = f(\mathbf{x}_i)$. Any prediction model may be selected as an underlying model as long as it presents results on an ordinal scale. In our case study we selected two count models, the Poisson regression model and the zero-inflated Poisson regression model.

### 2.3. Ranking Results

Let $\widehat{F}(\mathbf{x}_i)$ be an estimate of $F_i$ by the underlying model, $\widehat{f}(\mathbf{x}_i)$. $R_i$ is the notation for the perfect ranking of module $i$ according to $F_i$ while $\widehat{R}(\mathbf{x}_i)$ is the same ranking but according to $\widehat{F}(\mathbf{x}_i)$.

### 2.4. Performance Evaluation

The following evaluation method was previously developed at the Empirical Software Engineering Laboratory [2]. Given a model and a validation data set indexed by $i$:

1. Management will choose to enhance modules in a priority-based order, beginning with the most fault-prone. However, the rank of the last module enhanced is uncertain at the time of modeling. Based on the schedule and resources allocated for reliability enhancements activities, determine a range of percentiles that covers management's options for the last module to be enhanced. Choose a set of representative cutoff percentiles, $c$, from that range.
2. For each cutoff percentile value of interest, $c$, define the number of faults accounted for by modules above the percentile $c$:

$$G(c) = \sum_{i:R_i \geq c} F_i \tag{1}$$

$$\widehat{G}(c) = \sum_{i:\widehat{R}(\mathbf{x}_i)\geq c} F_i \qquad (2)$$

where a higher $c$ corresponds to the most fault-prone modules.

3. Let $G_{tot}$ be the total number of actual faults in the validation data set. For each ranking, compute the percentage of faults accounted for, i.e., $G(c)/G_{tot}$ and $\widehat{G}(c)/G_{tot}$, and depict the results of the model with an Alberg diagram [11]. Alberg diagrams are a variation of Pareto diagrams applied to module-order modeling.

4. Calculate a function measuring the model performance, $\phi(c)$, which indicates how closely the faults accounted for by the model ranking match with those of the perfect ranking.

$$\phi(c) = \frac{\widehat{G}(c)}{G(c)} \qquad (3)$$

We can then plot a performance graph described as a function of $c$.

## 3. Count Models

### 3.1. Count Models for Software Quality Modeling

Poisson [7] and zero-inflated Poisson are two regression models widely used for count data sets. A count data set is a set in which the response (dependent) variables are scalar non-negative integers. The response variable is the occurrence of the event. This applies very well to software quality modeling since most of the metrics are count metrics (we can hardly imagine a decimal or a negative number of faults).

### 3.2. The Poisson Regression Model (PRM)

The Poisson regression model (PRM) [7] is a regression model derived from the Poisson distribution. The Poisson regression model is part of a set of models for count data modeling. The best results for this model are obtained when the data set has a Poisson distribution. This implies that the expected count equals its variance. However, the Poisson regression model does not perform well when the data set contains a lot of zeros for the response variable.

#### 3.2.1 Poisson Distribution

The Poisson distribution is expressed as follow:

$$\Pr(y|\mu) = \frac{e^{-\mu}\mu^y}{y!} \qquad \text{for} \quad y = 0, 1, 2, \cdots . \qquad (4)$$

where $y$ is the random variable having a Poisson distribution with parameter $\mu$. For higher values of $\mu$ the behavior of the Poisson distribution approximates the behavior of a normal distribution. For the Poisson distribution, the expected value of $y$ is equal to its variance. This is also known as *equidispersion*:

$$\mathrm{E}(y) = \mathrm{Var}(y) = \mu \ . \qquad (5)$$

#### 3.2.2 Building the Poisson Regression Model

The Poisson regression model assumes that the response variable is a count and has a Poisson distribution with mean $\mu$, which is dependent on the covariates, $\mathbf{x}_i$. Let $(y_i, \mathbf{x}_i)$ be an observation in the data set. Given $\mathbf{x}_i$, assume $y_i$ has a Poisson distribution with a density function:

$$\Pr(y_i|\mu_i, \mathbf{x}_i) = \frac{e^{-\mu_i}\mu_i^{y_i}}{y_i!} \qquad \text{for} \quad y_i = 0, 1, 2, \cdots , \qquad (6)$$

where $\mu_i$ is the mean value of the response variable $y_i$. Since $\mu_i$ is always positive, the link function which demonstrates the relationship between the expected value of response variable and the covariates, generally has a logarithm form as shown below.

$$\ln(\mu_i) = \ln(\mathrm{E}(y_i|\mathbf{x}_i)) = \mathbf{x}_i'\beta \ , \qquad (7)$$

where ln means natural logarithm, $\mathbf{x}_i$ represents the covariates, $\mathbf{x}_i'$ is the transpose of the vector $\mathbf{x}_i$, and $\beta$ is a vector of the unknown parameters. Here, both $\mathbf{x}_i$ and $\beta$ are vertical vectors. The expected value $E(\cdot)$ is equivalent to the mean value denoted by $\mu$ throughout this paper. Sometimes, the link function is also written in an exponential form:

$$\mu_i = \mathrm{E}(y_i|\mathbf{x}_i) = e^{\mathbf{x}_i'\beta} \ . \qquad (8)$$

According to the equidispersion property (Equation (5)) of the Poisson distribution, it follows that

$$\mathrm{Var}(y_i|\mathbf{x}_i) = e^{\mathbf{x}_i'\beta} \ , \qquad (9)$$

Equation (6) and Equation (7) jointly define the Poisson regression model.

#### 3.2.3 PRM Prediction

In order to estimate the parameters for PRM, we use a standard technique called Maximum Likelihood Estimation [7]. When the model is built, one can utilize this model to predict the response (dependent) variable for each module. A prediction model for the PRM can be written as

$$\hat{y}_i = \hat{\mu}_i = e^{\mathbf{x}_i'\hat{\beta}} \qquad \text{for} \quad i = 1, 2, \cdots \ . \tag{10}$$

### 3.3. The Zero-Inflated Poisson Regression Model (ZIP)

As we already stated, it is often seen that a data set has excess zeros for the response variable in software quality modeling. A pure PRM is not suitable for this situation [1]. A zero-inflated count model is an effective way of dealing with this problem. A zero-inflated count model assumes that zeros can be generated by a different process than positive counts. Thus, this kind of model modifies the mean structure such that the conditional variance and the probability of zero count increase. Mullahy used a With-Zeros (WZ) model in his application [10]. In 1992, Lambert first introduced the zero-inflated Poisson regression model [9].

#### 3.3.1 Definition

In a zero-inflated model, we group all zeros into two parts. One part comes from the perfect modules, i.e., modules with zero faults. The other part comes from the non-perfect modules where the number of faults follows some standard distribution. In a ZIP regression model, we introduce a parameter $\psi$ which represents the probability of a module being perfect. Hence, the probability of the module being non-perfect is $1 - \psi$. Also, we assume that in non-perfect modules, the number of faults follows a Poisson distribution.

Let the response variable $\mathbf{y} = (y_1, y_2, \cdots, y_n)$ be independent and

$$\begin{aligned} y_i &= \textit{perfect} & &\text{probability} \ \ \psi_i \ , \\ &= \textit{non-perfect} \sim \text{Poisson}(\mu_i) & &\text{probability} \ \ 1 - \psi_i \ . \end{aligned}$$

The probability density function (pdf) of the ZIP regression model therefore is

$$Pr(y_i|\mathbf{x}_i, \psi_i) = \begin{cases} \psi_i + (1-\psi_i)e^{-\mu_i}, & y_i = 0, \\ (1-\psi_i)\frac{e^{-\mu_i}\mu_i^{y_i}}{y_i!}, & y_i = 1, 2, \cdots \ . \end{cases} \tag{11}$$

The ZIP regression model is obtained by adding the following link functions:

$$\ln(\mu_i) = \mathbf{x}_i'\beta \tag{12}$$

$$\text{logit}(\psi_i) = \ln \frac{\psi_i}{1-\psi_i} = \mathbf{x}_i'\beta \ , \tag{13}$$

where both $\beta$ and $\gamma$ represent the coefficient vectors of the covariates $\mathbf{x}_i$. However, $\beta$ serves the log function of mean $\mu_i$ and the $\gamma$ serves as the logit function of probability $\psi_i$.

#### 3.3.2 ZIP Regression Model Prediction

In order to estimate the parameters for ZIP, we again use Maximum Likelihood Estimation [7]. When the parameters $(\beta, \gamma)$ are estimated, one can utilize the ZIP regression model to predict the response variable for each module, and find the probability for each module being perfect. The probabilities of the response variable being various counts can be obtained by using Equation (11) of the ZIP regression model.

The mean prediction for the ZIP regression model is

$$E(y_i|\mathbf{x}_i) = (1 - \widehat{\psi}_i)e^{\mathbf{x}_i'\widehat{\beta}} \ . \tag{14}$$

where $\hat{\psi}_i$ is the predicted probability of module $i$ being perfect. $\hat{\psi}_i$ is estimated by the following:

$$\widehat{\psi}_i = \frac{e^{\mathbf{x}_i'\widehat{\gamma}}}{1 + e^{\mathbf{x}_i'\widehat{\gamma}}} \ . \tag{15}$$

## 4. Experiments

### 4.1. System Description

Our research was conducted on two data sets obtained from the industry. The data sets were both collected from the same project within the same organization. The project consisted of two large Windows-based applications used primarily for customizing the configuration of wireless products. The data sets were obtained from the initial releases. The applications are written in C++, and they both provide similar functionality. As a result, the applications contain common code. The main difference between the two applications is the type of wireless product that it can support. Table 1 presents the profile of the system used for this case study.

Table 2 lists the five software metrics used in our case study. The dependent variable, *Fault*, was the number of faults found in a module during system test. A software module was defined as a set of related source-code files.

The system had about only 33% of modules containing faults. The most faulty module had 97 faults. Figure 1 shows the distribution of *Fault*.

### 4.2. Experiments Description

We used data-splitting as a validation strategy. The *fit* data set was composed of 807 modules and the *test*

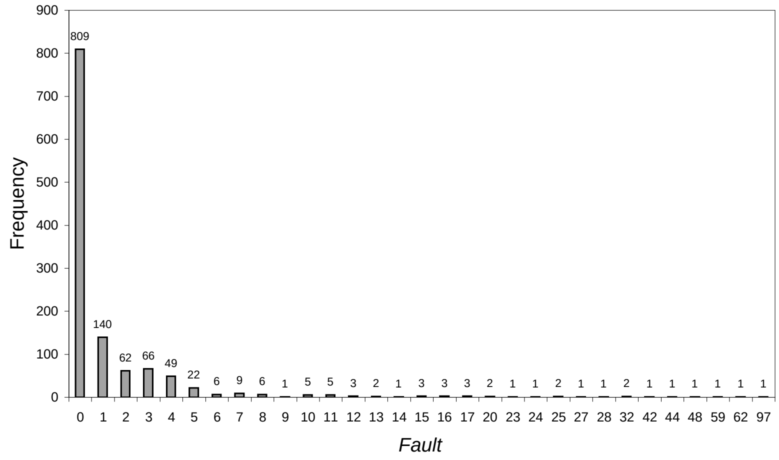


**Figure 1. Distribution of** *Fault*

**Table 1. System Profile for Case Studies**

| Applications | Service Configuration Software |
|---|---|
| Language | C++ |
| Application 1: AENSCL* | 320 million |
| Application 1: Actual Lines of Code | 29 million |
| Application 2: AENSCL* | 300 million |
| Application 2: Actual Lines of Code | 27.5 million |
| Number of source files | 1207 |

* AENCSL is Assembly Equivalent
Non-Commented Source Lines of Code

**Table 2. Software product metrics**

| Symbol | Description |
|---|---|
| *Ins* | Number of times the source file was inspected prior to system test release. |
| *BCode* | Number of lines of code for the source file prior to the coding phase. This represents auto-generated code. |
| *SCode* | Number of lines of code for the source file prior to system test release. |
| *Bcomm* | Number of lines of commented code for the source file prior to the coding phase. This represents auto-generated code. |
| *Scomm* | Number of lines of commented code for the source file prior to system test release. |

data set was composed of 404 modules. The *fit* data set was used to build the regression models and the *test* data set was used to evaluate the fitted model by computing the Average Absolute Error (AAE) and Average Relative Error (ARE) values. In order to provide independent (thus relevant) results we used 50 different data splits. This ensures that our final results are not an artifact of a lucky data split.

The results for module-order modeling were obtained using the Software Measurement Analysis and Reliability Toolkit (SMART) [4]. SMART is a research tool developed at the Empirical Software Engineering Laboratory, Florida Atlantic University.

## 4.3. Patterns Presentation

Once we completed our empirical investigation on all fifty data splits, we selected three distinct patterns. These patterns are presented in Table 3. Other data splits did not provide any additional information. We did not experience any data splits providing evidence of a better prediction accuracy of the Poisson regression model compared to the ZIP regression model.

Pattern 1, was selected because both count models had good prediction accuracy and ZIP provided a significant improvement over Poisson in terms of prediction. With Pattern 2, the two count models had an even better prediction accuracy but there was no significant improvement when using ZIP. Finally, Pattern 3 presents the data split where the predictions for both count models were extremely inaccurate, and for this type of data split, ZIP performed significantly "better" in terms of prediction. Table 4 presents descriptive statistics for the dependent variable, *Fault*, for each selected pattern.

**Table 3. Pattern Description**

| Pattern | Description | Selected Data Split |
|---|---|---|
| 1 | Good predictions for both count models. ZIP performs significantly better than Poisson. | 49 |
| 2 | Better prediction than Pattern 1 for both count models. The improvement of ZIP over Poisson is not significant. | 25 |
| 3 | Predictions for both count models are bad. ZIP performs significantly better than Poisson. | 20 |

### 4.4. Prediction Results

There are many ways to evaluate the quality of the prediction. Average Absolute Error (AAE) and Average Relative Error (ARE) [13] are the most common ways. The definition of AAE and ARE used in our study are:

$$AAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (16)$$

$$ARE = \frac{1}{n}\sum_{i=1}^{n} \left|\frac{y_i - \hat{y}_i}{y_i + 1}\right| \quad (17)$$

where $n$ is the number of modules in the test or evaluation data set, $y_i$ and $\hat{y}_i$ represent the actual and predicted value of the response (dependent) variable, respectively. In ARE, since the actual response variable (number of faults) may be zero, we add a one to the denominator to make the definition always well-defined [8]. Lower values of ARE and AAE indicate better prediction accuracy.

Table 5 presents in detail the prediction statistics of both count models for each pattern. The values reported are those of the data splits listed in Table 3 for each pattern respectively. The table also includes the $p$-value of the significance test between the two count models. The alternative hypothesis for the z-test was that the ZIP regression model had a better prediction accuracy than the Poisson regression model. The results allowed us to define the different patterns among the data splits.

Shepperd and Kadoda [12] mentioned that there is no *best* prediction model for software quality. The suc-

**Table 4. Descriptive Statistics of *Fault*, Y**

| | FIT (807) | | | | |
|---|---|---|---|---|---|
| Pattern | Number of Zeros | Total | | Positive | |
| | | Mean of $Y$ | Standard Deviation of $Y$ | Mean of $Y$ | Standard Deviation of $Y$ |
| 1 | 532 | 1.33 | 3.95 | 3.89 | 5.98 |
| 2 | 533 | 1.55 | 5.44 | 4.57 | 8.57 |
| 3 | 536 | 1.41 | 4.39 | 4.21 | 6.76 |

| | TEST (404) | | | | |
|---|---|---|---|---|---|
| Pattern | Number of Zeros | Total | | Positive | |
| | | Mean of $Y$ | Standard Deviation of $Y$ | Mean of $Y$ | Standard Deviation of $Y$ |
| 1 | 277 | 1.95 | 7.35 | 6.19 | 12.10 |
| 2 | 276 | 1.50 | 5.12 | 4.72 | 8.24 |
| 3 | 273 | 1.77 | 6.84 | 5.47 | 11.17 |

cess of a prediction technique has a strong relationship with the characteristic of the data set.

In our case study, the differences in terms of prediction accuracy are mostly explained by the effect of two extreme data points in our data set. These two extreme data points had a high influence on the parameters of the count models. In the case of Pattern 1 where both models had a good prediction accuracy and one influential point was in the *fit* data set while the other was in the *test* data set. The best predictions were obtained when both influential points were in the *fit* data set. This case is presented in Pattern 2. The worst predictions described in Pattern 3, were obtained when the two influential points were in the *test* data set. The large differences in prediction accuracy are explained by the fact that the parameters estimated were quite different in Pattern 3 because of the two influential points. However, a complete discussion on the impact of influential points on count models is out of the scope of this paper.

## 4.5. Module-Order Modeling Results

In this section, we present the module-order modeling results obtained using the quantitative prediction of the count models for the three selected patterns. The cutoff values varied from 5% to 35%. We used a 1% increment between 5% and 15% since they represent the most critical range for our case study. Cutoff points below 5% were not presented. Below the 5% cutoff, performances present a high variability due to the fact that we are ranking the most faulty modules. In addition this range is of little interest since we can hardly imagine inspecting less than 5% of the modules. However, ZIP usually presented better performances below the 5% cutoff.

The Alberg diagram will give a synthetic view of the advantages of module-order modeling. The model performance figure is directly extracted from the Alberg diagram information and presents the comparison between the two models. Section 2 of this paper presents module-order modeling in further details.

### 4.5.1 Pattern 1

Pattern 1 was selected because of its good prediction accuracy and the significant improvements of the ZIP regression model as compared to the Poisson regression model. In Figure 2 we can notice some important trends.

First of all, we can notice that the Pareto law (20% of a population is responsible for 80% of the phenomenon) applies to software quality modeling: 20% of

**Table 5. Prediction Results**

| | | Poisson | | | ZIP | | | *p*-value | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Pattern | Statistics | Total | Zero | NonZero | Total | Zero | NonZero | Total | Zero | NonZero |
| 1 | AAE | 2.9094 | 0.9199 | 7.2488 | 1.7752 | 0.8101 | 3.88 | 0.0901 | $< 0.0002$ | 0.1020 |
| | ARE | 0.8146 | 0.9199 | 0.5851 | 0.6994 | 0.8101 | 0.4579 | 0.0057 | $< 0.0002$ | 0.1539 |
| | std* AAE | 16.2191 | 0.151 | 28.5243 | 5.0274 | 0.4851 | 8.5913 | | | |
| | std* ARE | 0.7586 | 0.151 | 1.3089 | 0.5146 | 0.4851 | 0.4957 | | | |
| 2 | AAE | 1.5815 | 1.0043 | 2.8261 | 1.4483 | 0.8104 | 2.8236 | 0.2483 | $< 0.00002$ | 0.5000 |
| | ARE | 0.8011 | 1.0043 | 0.3629 | 0.6869 | 0.8104 | 0.4206 | 0.0000 | $< 0.00002$ | 0.9236 |
| | std* AAE | 2.8339 | 0.1031 | 4.814 | 2.7183 | 0.4204 | 4.5026 | | | |
| | std* ARE | 0.3536 | 0.1031 | 0.3008 | 0.4363 | 0.4204 | 0.3417 | | | |
| 3 | AAE | 39.1065 | 0.8715 | 118.787 | 7.186 | 0.7033 | 20.6957 | 0.1379 | $< 0.00002$ | 0.1379 |
| | ARE | 1.2702 | 0.8715 | 2.1011 | 0.6945 | 0.7033 | 0.6763 | 0.1075 | $< 0.00002$ | 0.1611 |
| | std* AAE | 579.967 | 0.1241 | 1016.4908 | 95.5581 | 0.5044 | 167.4351 | | | |
| | std* ARE | 9.2123 | 0.1241 | 16.1872 | 1.5831 | 0.5044 | 2.69 | | | |

* std is standard deviation.

the modules accounted for 93% of the faults and 35% of the modules accounted for 100% of the faults. We also observe that when using our module-order models we may achieve effective software quality enhancements. By reviewing 35% of the modules, we may catch 84% of the faults using our count models for module-order modeling. Finally, we may notice that the two count models although being significantly different in terms of prediction, performed similar with respect to their subsequent module-order models.

Model performance (Figure 3) gives a closer view to the comparative behavior of the models over the selected cutoff points. We notice that the maximum difference between the two models is about 3% with a very high overall performance.

#### 4.5.2 Pattern 2

For Pattern 2, both count models had very good prediction accuracy. The prediction accuracy was significantly better for both models compared to Pattern 1. However, there was no significant improvement when using ZIP instead of Poisson. Figure 5 indicates that the two models remain close to each other (within 8%) with ZIP performing similar or slightly better than Poisson. The most important point here is that although both models had a better prediction accuracy (as compared to Pattern 1), their module-order model performance was not as good as that of Pattern 1 (see Figure 4).

#### 4.5.3 Pattern 3

This pattern was chosen to show the module-order modeling results when the predictions for the count models had very low accuracy. For this pattern, ZIP performed significantly "better" in terms of prediction.

Figure 6 and Figure 7 show that although the prediction accuracy of both models was extremely poor their module-order model performances remained excellent. The maximum difference between the performances of the two models did not exceed 4%. The module-order modeling performances for Pattern 3 were similar to the performances for Pattern 1 and were better than Pattern 2 despite having extremely poor prediction accuracy.

## 5. Conclusion and Future Work

Software quality models can be used to predict a quality factor using software metrics collected early in the life cycle. Often predicting the exact value of the quality factor is not sufficient. Instead, predicting the rank-order of modules with respect to the quality factor may be more beneficial. Managers can then target the modules with the highest ranking until they exhaust allowable resources.

A module-order model predicts this ranking according to a quality factor, such as the number of faults. The quantitative prediction of the quality factor is made by an underlying quantitative prediction model.

This paper demonstrated that improving a software quality model for prediction did not yield the same significant improvement for the module-order modeling performances. In addition we showed that the best prediction did not yield the best module-order modeling results and that having a poor prediction did not seem to be a handicap for module-order modeling.

The case study of a full-scale industrial software system compared the performances of two different count models with the following conclusions.

- Prediction accuracy indicators such as AAE and ARE do not give a reliable indication of the future behavior of the prediction model when used as a module-order model.
- Improved models for quantitative prediction, like the zero-inflated Poisson regression model compared to the Poisson regression model do not automatically yield significant module-order modeling performance improvements.
- A better prediction accuracy does not necessarily yield better module-order modeling performances.
- Bad and extremely bad prediction accuracy do not seem to significantly affect the module-order modeling performances in one way or another.
- Module-order models are very robust. They provide very similar performances regardless of the prediction accuracy of the underlying prediction models.
- The choice of the underlying model should be determined empirically according to the chosen cutoff and the characteristics of the data set.
- Managerial issues such as process improvements and simplicity of the models can help choosing an adequate prediction model.

Overall, we found that module-order modeling remains a very efficient technique for both count models and that the choice of an underlying model should not solely rely on the prediction accuracy.

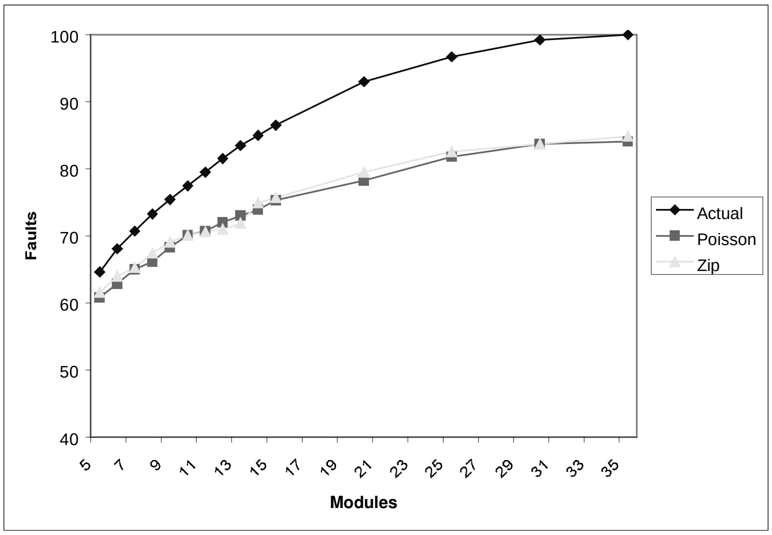


**Figure 2. Alberg Diagram for Pattern 1.**

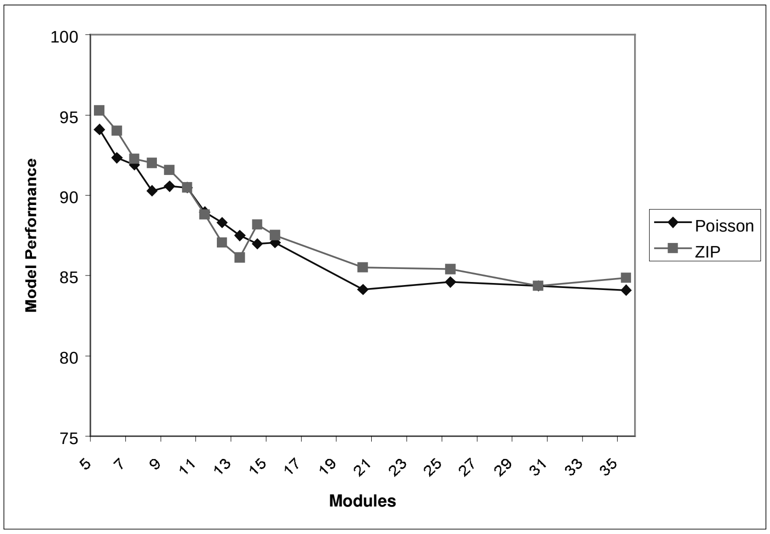


**Figure 3. Performance for Pattern 1, $\phi(c)$.**

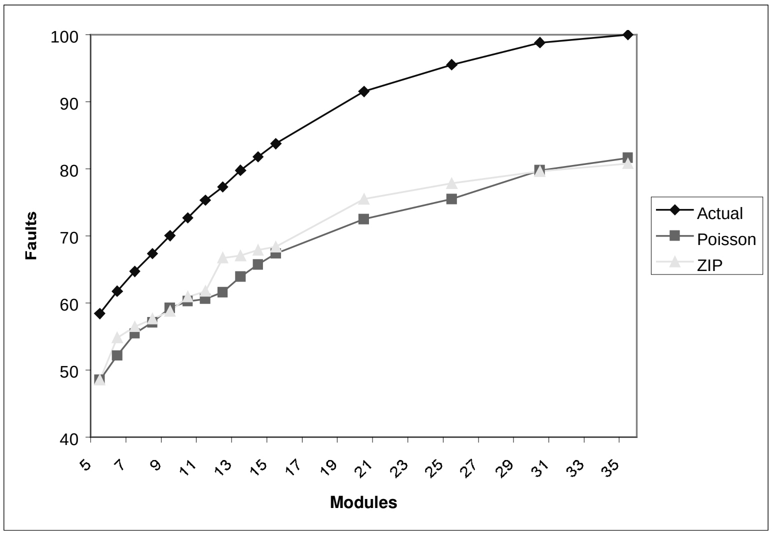


**Figure 4. Alberg Diagram for Pattern 2.**

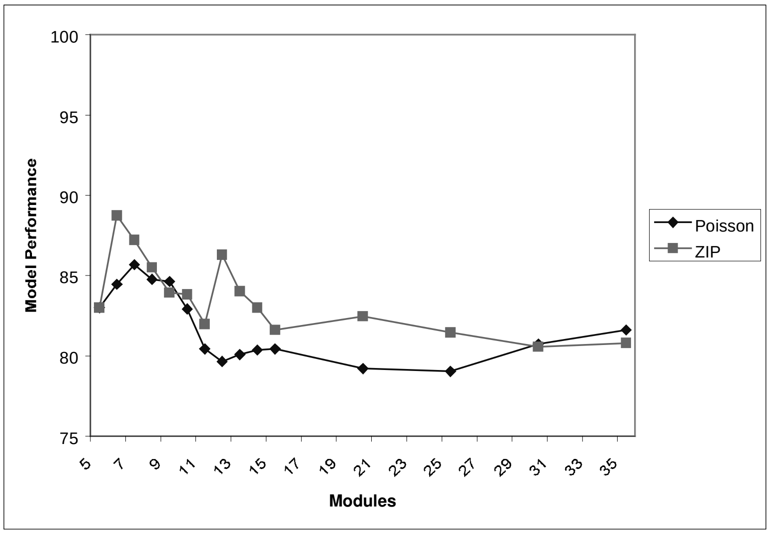


**Figure 5. Performance for Pattern 2, $\phi(c)$.**

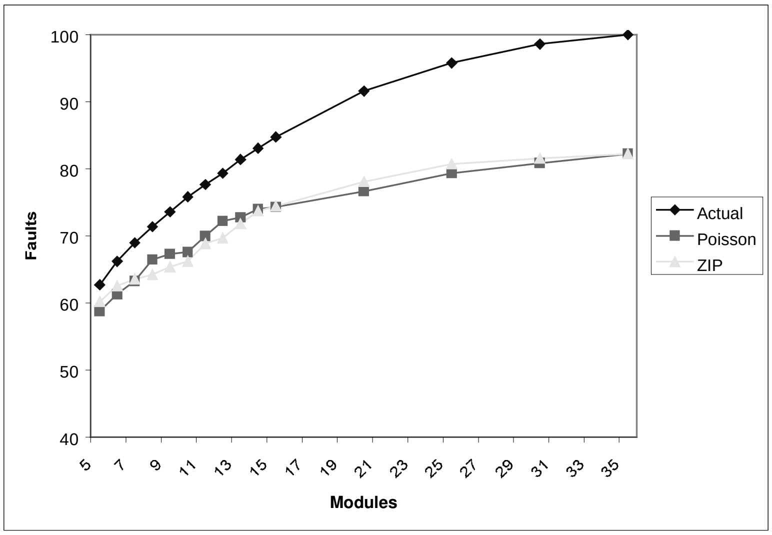


**Figure 6. Alberg Diagram for Pattern 3.**

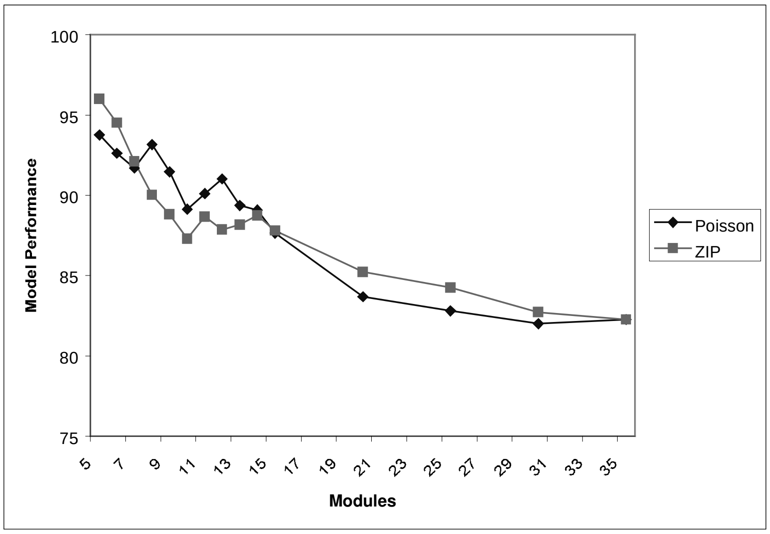


**Figure 7. Performance for Pattern 3, $\phi(c)$.**

Future work should investigate better indicators than AAE and ARE to help to build the best underlying model from a set of candidates for module-order modeling.

## Acknowledgments

We thank Ken McGill for his encouragement and Dr. Bojan Cukic for his helpful suggestions. This work was supported in part by Cooperative Agreement NCC 2-1141 from NASA Ames Research Center, Software Technology Division (Independent Verification and Validation Facility). The findings and opinions in this paper belong solely to the authors, and are not necessarily those of the sponsor. Special thanks to Naeem Seliya for his patient reviews.

## References

[1] W. H. Green. Accounting for excess zeros and sample selection in poisson and negative binomial regression models. Technical Report EC-94-10, Economics Department, New York University, 1994.

[2] T. M. Khoshgoftaar and E. B. Allen. A comparative study of ordering and classification of fault-prone software modules. *Empirical Software Engineering: An International Journal*, 4:159–186, 1999.

[3] T. M. Khoshgoftaar and E. B. Allen. Modeling software quality with classification trees. In H. Pham, editor, *Recent Advances in Reliability and Quality Engineering*, pages 247–270. World Scientific, Singapore, 2001.

[4] T. M. Khoshgoftaar, E. B. Allen, and J. C. Busboom. Software quality modeling: The software measurement analysis and reliability toolkit. In *Proceedings of the Twelfth IEEE International Conference on Tools with Artificial Intelligence*, pages 54–61, Nov. 2000.

[5] T. M. Khoshgoftaar, M. P. Evett, E. B. Allen, and P.-D. Chien. An application of genetic programming to software quality prediction. In W. Pedrycz and J. F. Peters, editors, *Computational Intelligence in Software Engineering*, volume 16 of *Advances in Fuzzy Systems — Applications and Theory*, pages 176–195 . World Scientific, Singapore, 1998.

[6] T. M. Khoshgoftaar, K. Ganesan, E. B. Allen, F. D. Ross, R. Munikoti, N. Goel, and A. Nandi. Predicting fault-prone modules with case-based reasoning. In *Proceedings of the Eighth International Symposium on Software Reliability Engineering*, pages 27–35, Albuquerque, NM USA, Nov. 1997. IEEE Computer Society.

[7] T. M. Khoshgoftaar, K. Gao, and R. M. Szabo. An application of zero-inflated poisson regression for software fault prediction. In *Proceedings: The Twelth International Symposium on Software Reliability Engineering*, Hong Kong, Nov. 2001. In press.

[8] T. M. Khoshgoftaar, J. C. Munson, B. B. Bhattacharya, and G. D. Richardson. Predictive modeling techniques of software quality from software measures. *IEEE Transactions on Software Engineering*, 18(11):979–987, Nov. 1992.

[9] D. Lambert. Zero-inflated poisson regression, with an application to defects in manufacturing. *Technometrics*, 34(1):1–14, Feb. 1992.

[10] J. Mullahy. Specification and testing of some modified count data models. *Journal of Econometrics*, 33:341–365, 1986.

[11] N. Ohlsson and H. Alberg. Predicting fault-prone software modules in telephone switches. *IEEE Transaction on Software Engineering*, 22(12):886–894, 1996.

[12] M. Shepperd and G. Kadoda. Using simulation to evaluate prediction techniques. In *Proceedings: Seventh International Software Metrics Symposium*, pages 349–359, London, England, Apr. 2001. IEEE Computer Society.

[13] R. M. Szabo and T. M. Khoshgoftaar. Exploring a poisson regression fault model: A comparative study. Technical Report TR-CSE-00-56, Florida Atlantic University, 2000.

[14] Z. Xu, T. M. Khoshgoftaar, and E. B. Allen. Application of fuzzy linear regression model for predicting program faults. In H. Pham and M.-W. Lu, editors, *Proceedings: Sixth ISSAT International Conference on Reliability and Quality in Design*, pages 96–101, Orlando, Florida USA, Aug. 2000. International Society of Science and Applied Technologies.