Predicting Post-release Defects Using Pre-release Field Testing Results

Foutse Khomh[1], Brian Chan[1], Ying Zou[1], Anand Sinha[2], and Dave Dietz[2]

[1] Dept. of Elec. and Comp. Engineering, Queen's University, Kingston, Ontario, Canada
[2] Handheld Software, Research In Motion (RIM), Waterloo, Ontario, Canada

E-mail: {foutse.khomh, 2byc, ying.zou}@queensu.ca

***Abstract*—Field testing is commonly used to detect faults after the in-house (*e.g.*, alpha) testing of an application is completed. In the field testing, the application is instrumented and used under normal conditions. The occurrences of failures are reported. Developers can analyze and fix the reported failures before the application is released to the market. In the current practice, the Mean Time Between Failures (MTBF) and the Average usage Time (AVT) are metrics that are frequently used to gauge the reliability of the application. However, MTBF and AVT cannot capture the whole pattern of failure occurrences in the field testing of an application. In this paper, we propose three metrics that capture three additional patterns of failure occurrences: the average length of usage time before the occurrence of the first failure, the spread of failures to the majority of users, and the daily rates of failures. In our case study, we use data derived from the pre-release field testing of 18 versions of a large enterprise software for mobile applications to predict the number of post-release defects for up to two years in advance. We demonstrate that the three metrics complement the traditional MTBF and AVT metrics. The proposed metrics can predict the number of post-release defects in a shorter time frame than MTBF and AVT.**

***Keywords*-Software reliability, metrics, prediction, post-release defects**

I. Introduction

As a software organization prepares to develop a new software application, they must plan the testing activities and the allocation of support resources. Such planning efforts play a central role in the timing of delivering and have an important impact on the cost of maintaining the application once released. Pre-release field testing, commonly referred to as beta testing, is performed when a new version of an application is near completion. Pre-release field testing verifies the use of the application by users in several real-life scenarios and under different configurations. The pre-release field testing period lasts for a few weeks. During this period, instrumented versions of the application are provided to a selected group of ordinary users who are not professional testers. The instrumented versions record problems encountered by users and report the problems to the development team. The reported problems include crashes and hangs in the application. A problem report contains information such as the time when the problem occurred, the user, and the stack trace snapshot from when the problem occurred. Because of the reported problems, developers are able to improve the reliability of the application and prepare for the release to the entire customer base.

In the current state of the practice, the Mean Time Between Failures (MTBF) and the Average usage Time (AVT) of the application during the field testing are used to evaluate the reliability of the application. More specifically, MTBF represents the average amount of time that passes between random failures of an application during operation [1]. Therefore, MTBF derived from the pre-release field testing period of an application can be used to predict the number of post-release defects. Applications with a low MTBF are undesirable. These applications would have a higher number of defects, which leads to a higher number of support calls and lower customer satisfaction [2]. AVT is the average time that a user actively uses the application. The AVT can be longer than the period of field testing, since the users continue to use the application even after the field testing. Therefore, a longer AVT indicates that an application is reliable and a user tends to use the application longer.

We believe that MTBF metrics and AVT fail to show the full picture of reliability during the pre-release field testing period. For example, if an application produces a large number of defects during the first two days of its pre-release usage, then users are likely to reduce their usage of the application. The reduced usage would result in the application being tested by a smaller number of users which in turn would lead to the occurrence of more defects once the application is released. Moreover, early estimates of the number of post-release defects are desirable as it helps to plan testing and allocation of support resources. We conjecture that the whole pattern of occurrences of failures during the pre-release field testing should be studied and quantified because it provides more information about the reliability and the quality of an application.

In this paper, we propose three metrics to quantify three major patterns of failure occurrences during the pre-release field testing of an application: the average length of usage time before the occurrence of the first failure, the spread of failures to the majority of users, and the daily rates of failures. Such patterns are not captured by the existing MTBF and AVT metrics. The proposed metrics are the first

to evaluate the three patterns. The purpose of the study of patterns of failures is to help improve the prediction of the number of post-release defects. Using statistical models and data from the pre-release field testing period of 18 versions of a large enterprise software for mobile applications, we study the independency between our proposed metrics to verify if they convey different aspects of reliability. We also use our metrics to predict the number of future post-release defects.

The main contribution of this paper is three proposed metrics that:

- complement the traditional MTBF and AVT in predicting the number of post-release defects; and
- provide faster predictions of the number of post-release defects with good precision within just 5 days of a pre-release testing period. In contrast, it takes MTBF up to 25 days to predict the number of post-release defects.

The rest of the paper is organized as follows. Section II motivates and presents our three metrics. Section III presents the design of our case study, reports its results, and discusses the threats to validity. Section IV discusses the related literature on software reliability and defect predictions. Finally, Section V concludes the paper and outlines future work.

II. Proposed Metrics

To capture the patterns of failures occurring in the field testing, we propose the following three metrics.

A. Average Time To First Failure (TTFF)

The Average Time To First Failure (TTFF) metric measures the average amount of time before a user faces his or her first failure. It is desirable for a user to use an application for a long period of time before the first failure occurs. With the first failure occurring early, users are likely to be more concerned about the quality and reliability of the application. This concern would in turn affect their usage patterns of the application and would impact the usefulness of the testing period with users not using the application as much. We define the TTFF metric in Equation (1).

$$TTFF = \sum_{i=1}^{T} i * P_i \quad (1)$$

Where T represents the length of the testing period in days; i represents the $i-th$ day of using the application in the testing period; and P_i refers to the percentage of users that reported their first failure on the $i-th$ day.

The TTFF metric accounts for the first failure for each user across all field testing users. Subsequent failures for the same user are not considered. Figure 1 shows an example of the TTFF metric calculation. The x-axis represents the days of the field testing period. The y-axis is the percentage of users reporting their first failure on a specific day. An application would exhibit a low score if it reports most of its first failures in the early days of the testing period. For example, the version B trend in Figure 1 illustrates high values of P_i (*i.e.*, the percentage of users that report their first failure on day i) during the early days in the testing period. This produces a low Time To First Failure score since the number of days, i, in Equation (1) creates low values for early days. However, version A has a more distributed trend with a low percentage of users reporting the first failure throughout the testing period. The P_is of version A are lower than those of version B in the beginning but they are consistently higher in the later stages of the testing period when the number of days progressively increases. Therefore, version A produces a higher TTFF score. It indicates that more users experience no failures or the first failure in a later time. The TTFF metric produces high scores for applications where the majority of users experience the first failure late. Therefore, a high score is desirable.

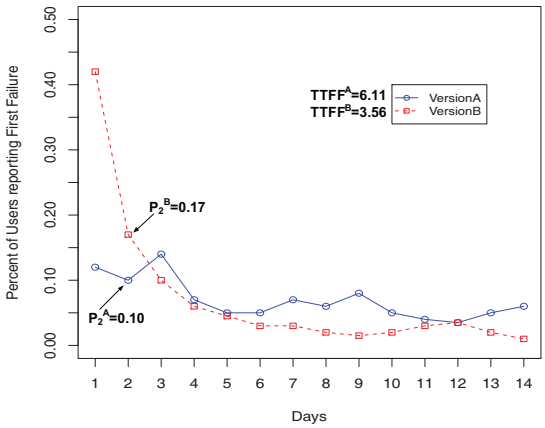

Figure 1. Sample Data Input for the Average Time To First Failure Metric

B. Failure Accumulation Rating (FAR)

The Failure Accumulation Rating (FAR) metric analyzes the distribution of failures to identify where the majority of users fall in terms of the total number of failures. Figure 2 depicts an example graph to show the distribution of the majority of the users. The x-axis represents the number of unique failures, which records only the first occurrence of a failure and ignores the subsequent occurrences of the same failure. The y-axis shows the cumulative percentage of users that report a certain number of unique failures for the entire testing period. P_i corresponds to the cumulative percentage of users who encountered at most i unique failures. The version B trend for the example shows that 22% of the users report 0 failures during the entire testing period (*i.e.*, P_0=0.22), while for the version A trend, 82% of the users report 0 failures during the entire testing period. As shown in Figure 2, the reliability of these two applications is very different.

A high percentage of users reporting a small number of unique failures is most desirable. To capture the overall failure distribution and its impact on the majority of users,

we assign a weight to the different number of unique failures that the users encountered. A low number of failures is assigned a high weight since a low number of failures is more desirable. Therefore, the Failure Accumulation Rating produces higher scores for applications where the majority of users report low numbers of failures. We define the FAR metric in Equation (2).

$$FAR = \sum_{i=0}^{N_F} P_i * (1 - \frac{i}{N_F + 1}) \quad (2)$$

Where i represents the number of unique failures reported by a user, N_F is the maximum number of unique failures reported by a user (i.e., i ranges from 0 to N_F; the total number of the possible values for i is therefore $N_F + 1$); and P_i is the cumulative percentage of users who encounter at most i unique failures.

With FAR, an application with most of the users reporting a low number of failures (*i.e.*, high P_i for small i) exhibits a higher score compared to an application with users reporting a higher number of failures. For example, the version A trend in Figure 2 shows that 82% of users report 0 failures. This produces a high Failure Accumulation Rating score since the coefficient, $1 - \frac{i}{N_F+1}$, shown in Equation (2) creates high values for low numbers of failures. The value of the coefficient is 1 for 0 failures, and decreases to its lowest, $\frac{1}{N_F+1}$, for the maximum number of failures, *i.e.*, N_F. The version A shows a desirable trend because most of its users report low numbers of failures. The FAR score for version A is 6.97. Conversely, version B shown in Figure 2 has more users reporting a large number of failures. Only 22% of its users report 0 failures. This trend indicates a poor reliability, which is successfully captured by the low FAR score of 4.97 for version B. The FAR metric produces high scores for applications where the majority of users report a very low numbers of failures.

The FAR metric is designed to capture the distribution of failures encountered by users of an application instead of just the mean or median. In addition, we found through an experiment that the mean and median of numbers of unique failures encountered by a user have a Spearman correlation of respectively 0.7 and 0.8 with TTFF. Therefore, a metric based on the mean or the median would not capture new information.

C. Overall Failure Rating (OFR)

The Overall Failure Rating (OFR) captures the distribution of the percentage of users that report failures each day during the testing period. For example, Figure 3 shows two applications displaying different trends of daily percentages of users reporting failures. The x-axis tracks the days over a field testing period. The y-axis represents the percentage of users that report unique failures in a given day. It is desirable that a low percentage of users report failures every testing

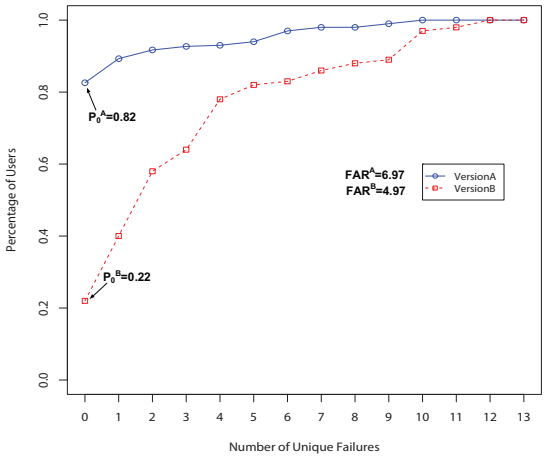

Figure 2. Sample Data Input for Failure Accumulation Rating Metric

day. Indeed, a lower percentage of users reporting unique failures daily indicates a better overall software reliability. Therefore, similarly to Keene's use of the exponential of failure rates to assess a system reliability [3], we compute the failure rating of each day i using the exponential of P_i, (*i.e.*, e^{-P_i}). Using this equation, the low percentage of failures per day produces a higher value than the high percentage of failures per day.

For each day, an ideal case is when an application has 0% of the users reporting failures. This produces the highest failure rating (*i.e.*, $e^{-0} = 1$). The worst case is when all the users (*i.e.*, 100%) report failures. The failure rating is then $e^{-1} = \frac{1}{e}$, which is the lowest.

We define the Overall Failure Rating (OFR) to be the average of daily failure ratings throughout the testing period. The definition is specified in Equation (3):

$$OFR = \sum_{i=1}^{T} \frac{e^{-P_i}}{T} \quad (3)$$

Where T represents the length of the testing period in days; i represents the $i-th$ day of using the application during the testing period; and P_i refers to the percentage of users that report at least one failure on the $i-th$ day.

For the example shown in Figure 3, version A displays a desirable trend with a lower percentage of users reporting unique failures throughout the testing period than version B. Version A has an OFR value of 0.93, whereas version B has an OFR value of 0.78. The trend of version B indicates a poor reliability, which is successfully captured by the low OFR score. The OFR metric produces high scores for applications with fewer users reporting failures overall.

III. Case Study

The *goals* of this case study are to:

1) investigate the independency among the proposed metrics; and
2) examine the predictive power of the proposed metrics for post-release defects.

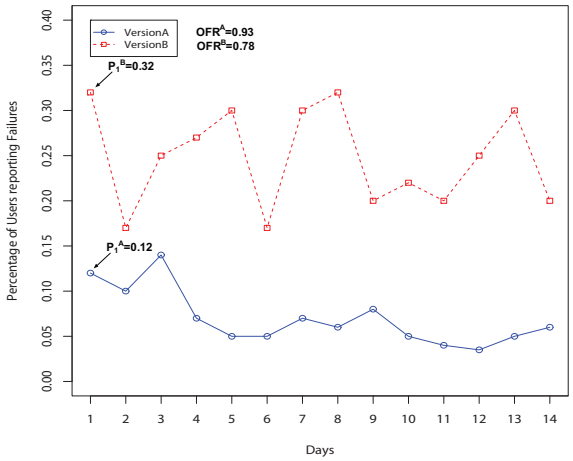

Figure 3. Sample Data Input for Overall Failure Rating Metric

The *motivation* of this case study is the prediction of the number of post-release defects of an application. A high number of post-release defects indicates a poor reliability for the application, and can have an important impact on the cost of maintaining the application once released. Managers and quality assurance personnel can use our metrics to predict the number of post-release defects of the applications to be released, and better plan the testing activities and the allocation of support resources to reduce maintenance costs.

The *context* of this study consists of data derived from the field testing of 18 versions of a large enterprise software system for mobile applications, collected during the past four years. 2,767 users participated in the field testing of these versions. We use data for the first 30 days of the field testing phase since our analysis of the growth rate of unique failures plateaus at around 30 days for most software versions. We also mined the bug report repository to obtain the number of post-release defects reported for up to two years after the application is released.

In the following subsections, we describe the challenges associated with collecting the data needed for our study. We present our research questions, and describe our analysis method. Then we present and discuss the results of our study as well as the threats to their validity.

A. Data Collection Challenges

During the field testing, a specialized usage tracking agent is installed. The agent reports various aspects about the usage of the application as well as any failures. The reported information is archived in a field testing repository. We mined this repository to recover the information needed for our case study. We had to address a number of challenges to prepare the data for further analysis. Below we present a brief discussion of these challenges:

Duplicate failures. Within a very short period of time, users might re-attempt the same operation that caused the preceding failure. This would lead to repeated failures in the repository. A similar phenomenon was noted by Bettenburg et al. [4] for the Mozilla open source project. To overcome this challenge, we check the timestamp of the failure and the reported stack trace in the failure report. If the stack traces are identical and the failures are reported within one hour apart, we do not count the second failure.

Participation of users. During the field testing period, users are likely to join or drop out at any time. However, all of our metrics assume that users use the application in a consistent manner. We developed several techniques to capture the participation of a user in the field testing.

Given that users can join the field testing period at any time, we mine the field reports in the repository to adjust the usage times for all users. More specifically, we put the starting dates of all users to a universal Day 1 and increment the Day with every new day of usage by the user. The days of usage are determined through the field reports submitted by the tracking agent.

Since users can temporarily or permanently stop using an application, we need to develop techniques to determine the exact number of days of usage. We use other types of field reports to identify the usage of the application. For example, the tracking agent reports the usage of other features and applications. We use these reports to determine if a particular user is still active even if we have not received usage updates from the specific application. All data evaluated in this case study are based on a period of 30 days. We also track the usage reports across the different versions. For example, if a user stops reporting the use of a particular version of the application, we go through our repository to determine if the user might have upgraded to a different version and is now part of the field testing of another version.

B. Research Questions

We address the following research questions within the context of our case study:

1) **RQ1:** Do metrics TTFF, FAR, OFR, AVT, and MTBF demonstrate an independence of factors?
2) **RQ2:** How early in the field testing cycle can TTFF, FAR, and OFR metrics predict the number of post-release defects using the fielding testing data?
3) **RQ3:** Are TTFF, FAR, and OFR more important than existing metrics AVT and MTBF for predicting the number of post-release defects?

To address our research questions, we compare TTFF, FAR, and OFR to the traditional MTBF and AVT metrics. MTBF computes the average failure-free period of a user in between failure events. AVT computes the average time that a user actively participated in the field testing before switching to another version. Large positive values of MTBF indicate that after one failure, a user is likely to enjoy a long period of time before encountering another failure. We compare the proposed metrics against MTBF since it is one of the most widely used metrics to measure software reliability [1], [2]. We also compare against AVT. We expect

that the longer the time period that users use the application, the less likely they would have encountered problems that discourage them from using the version.

C. Analysis Method

RQ1. We study whether the three metrics TTFF, FAR, and OFR convey different information about the reliability of a software application compared to MTBF and AVT. To verify that our proposed metrics are essentially different from MTBF and AVT, we measure the correlations among all the metrics. We perform a Principal Component Analysis (PCA) to understand the underlying, orthogonal dimensions captured by the metrics TTFF, FAR, OFR, and MTBF. This method is similar to previous work [5], [6]. We do not include AVT in the PCA because of its high variance and high values, compared to the other metrics.

We use the non-parametric Spearman correlation [7] which is suitable when the number of subjects is not very high. This is the case in our study, since we have 18 versions. The strength of the correlation shows the similarity among the metric results. A weak or no correlation among the results of metrics is desirable. It indicates less dependence among the metrics, and in turn shows that the metrics convey different aspects of reliability. By performing PCA, we can identify groups of variables (*i.e.*, metrics), which are likely to measure the same underlying dimension (*i.e.*, specific aspect of reliability). It is desirable to have all the metrics to belong to orthogonal dimensions. We apply the same "varimax" rotation technique as Marcus *et al.* [5] to enhance the interpretability of the principal components. As the interpretability of the second and higher PCA components is typically limited, "varimax" rotates PCA components to maximize the variance of their loadings. This maximization makes each component as distinctive as possible compared to the other components. We use the scree test to determine the number of components to retain in the PCA. The scree test involves plotting the eigenvalues in descending order of their magnitude against their factor numbers. We use the break between the steep slope of the curve and a leveling off to identify the number of meaningful components.

RQ2 and RQ3. We want to understand to what extent the metrics TTFF, FAR, and OFR are able to predict the number of post-release defects of an application. We use a negative binomial regression model [8] to relate the number of post-release defects to a linear combination of the metrics. Unlike linear regression, a negative binomial regression model is able to handle an outcome that is a count. Negative binomial regression models explicitly model outcomes that are non-negative integers. In our case of post-release defects, the model assumes that the expected number of post-release defects varies as a function of the metrics in a multiplicative relationship.

Let y_i be the number of post-release defects of a version i and x_i be a vector of metrics for that version. The negative binomial regression model specifies that for an outcome Y, the probabilities of observing y_i, given x_i, has a Poisson distribution with mean λ_i [9], as in Equations (4), and (5).

$$Prob[Y = y_i|x_i] = \frac{\lambda_i^{y_i} e^{-\lambda_i}}{y_i!} \tag{4}$$

$$\lambda_i = \gamma_i e^{\beta x_i} \tag{5}$$

Where β is a vector of regression coefficients conforming to x_i, and γ_i is a random variable drawn from a gamma distribution with mean 1 and unknown variance $\sigma^2 \geq 0$. The variance σ^2 is known as the dispersion parameter. The regression coefficients β and the dispersion parameter σ^2 are estimated using the maximum likelihood [8]. We perform a Likelihood-ratio test to compare the model with metrics to the null model. We compute the p-value to assess the global significance of the model. We also compute McFadden's R^2 to measure the goodness of fit of the model. McFadden's R^2 values of 0.2 to 0.4 are considered highly satisfactory [10], meaning that the fitted model explains a high proportion of variability in y_i. To verify the relative predictive abilities of the metrics, we compute the marginal McFadden's R^2 of each metric. The marginal R^2 measures the predictive power of a variable. For each of the five metrics, we compute the corresponding marginal R^2 based on the likelihood ratio of the model with five metrics and the model with the remaining four metrics. A value of marginal R^2 shows the fraction of the variance that is explained by a metric. The higher the marginal R^2 of a metric, the higher the predictive power of that metric. A marginal R^2 value of 0 means that the metric has no contribution to the model.

We compute the precision of the models using a "leave-one-out cross-validation" approach. In each iteration, we used 17 out of the 18 versions of our enterprise application to train a negative binomial model that predicts the number of post-release defects in the remaining version. We repeat the process 18 times. We compute the precision achieved by each model, following the Information Retrieval (IR) metric defined in Equation (6) [11]:

$$precision = \frac{|PredictedCorrect|}{|Predicted|} \tag{6}$$

Where $PredictedCorrect$ represents the set of versions for which the correct number of post-release defects is predicted and $Predicted$ is the set of all versions for which the number of post-release defects is predicted.

For models that predict a binary classification (*i.e.*, T or F), the difference between precision and recall is that precision is relative to the number of observations predicted to be T, whereas recall is relative to the number of all observations that are T. Since our models do not predict a classification (*i.e.*, T or F), but a count (*i.e.*, the number of post-release defects), the set of observations for which we predict a count is the same as the set of all observations, *i.e.*, the concepts of precision and recall are identical.

D. Study Results

This section reports and discusses the results of our study.

RQ1: Do metrics TTFF, FAR, OFR, AVT, and MTBF demonstrate an independence of factors?

For each release, we measure the proposed metrics and MTBF using data for 30 days of field testing. Table I shows the results of the correlation between the metrics. The p values for all the correlations are less than 0.05. Overall, the correlations between the metrics are weak. A weak correlation between two metrics occurs when the correlation score is less than 0.4 [7]. AVT has a weak correlation of approximately 0.3 with TTFF, FAR, OFR, and MTBF. The only metrics with a moderate correlation (*i.e.*, 0.54) are OFR and MTBF. However, the PCA analysis results as listed in Table II suggest that each metric TTFF, FAR, OFR, and MTBF captures a unique dimension in the data, with TTFF alone capturing 91.9% of the variance in the data set. The results are good indicators that the five metrics TTFF, MTBF, FAR, OFR, and AVT, capture different aspects of an application reliability.

Table I
SPEARMAN CORRELATION RESULTS BETWEEN METRICS

	TTFF	FAR	OFR	AVT	MTBF
TTFF	1.00	0.09	-0.08	-0.31	-0.08
FAR	0.09	1.00	0.07	0.33	-0.24
OFR	-0.08	0.07	1.00	0.39	-0.54
AVT	-0.31	0.33	0.39	1.00	-0.30
MTBF	-0.08	-0.24	-0.54	-0.30	1.00

TTFF: Time To First Failure; FAR: Failure Accumulation Rating; OFR: Overall Failure Rating; AVT: Average Time that a user used the application; MTBF: Mean Time Between Failures.

Table II
RESULTS OF PRINCIPAL COMPONENT ANALYSIS

	PC1	PC2	PC3	PC4
Proportion	91.9%	6.38%	1.72%	0.02%
Cumulative	91.9%	98.26%	99.98%	100%
TTFF	**-0.99**	0.04	-0.01	0.00
FAR	0.01	0.13	**-0.98**	0.06
OFR	0.00	0.01	0.06	**0.99**
MTBF	-0.04	**-0.99**	-0.13	0.01

RQ2: How early in the field testing cycle can TTFF, FAR, and OFR metrics predict the number of post-release defects using the fielding testing data?

In this research question, we are interested in examining how early in the field testing cycle our proposed metrics can predict the number of post-release defects. If our proposed metrics can show good ability to predict the number of post-release defects early in the field testing period, then they are worth adopting to complement traditional metrics such as AVT and MTBF, which need a long period of testing to be effective predictors of the number of post-release defects. By closely monitoring our metrics throughout the field testing period, managers and quality assurance personnel can react faster to software quality problems instead of waiting until the testing period is completed. They can also decide to reduce the length of the testing period.

Table IV
SUMMARY OF RESULTS

Metrics	Minimum number of testing days necessary to predict		
	Six month post-release defects	One year post-release defects	Two years post-release defects
TTFF	–	5	–
FAR	–	5	20
OFR	20	5	10
AVT	–	30	–
MTBF	–	25	–

Table V
PRECISION, LIKELIHOOD-RATIO TEST, AND MCFADDEN'S R^2 RESULTS (SIGNIFICANT p-VALUES ARE HIGHLIGHTED IN BOLD)

Number of testing days	5	10	15	20	25	30	Average
Six months post-release defects							
Precision	87%	86%	87%	87%	87%	86%	86%
p-value	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	–
R^2	0.35	0.34	0.34	0.35	0.35	0.35	0.34
One year post-release defects							
Precision	58%	75%	56%	75%	60%	75%	66%
p-value	**0.05**	0.1	0.07	0.1	0.06	0.09	–
R^2	0.16	0.10	0.14	0.12	0.15	0.14	0.13
Two years post-release defects							
Precision	72%	73%	73%	80%	72%	73%	73%
p-value	**0.01**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	–
R^2	0.22	0.26	0.27	0.30	0.28	0.30	0.27

We compute TTFF, FAR, OFR, and MTBF using field testing data for 5, 10, 15, 20, 25, and 30 days. We compute AVT only for 30 days since it tracks all users and versions of the application during the entire usage period which can be longer than the field testing period.

Table III reports the coefficients of the negative binomial regression models built with field testing data from 5, 10, 15, 20, 25, and 30 days. Significant p-values and meaningful contributions of metrics (*i.e.*, $R^2 > 0$) are highlighted in bold. Table IV summarizes the shortest number of testing days needed for each metric to predict the number of post-release defects. Table V summarizes the precision achieved using data from 5, 10, 15, 20, 25, and 30 days. McFadden's R^2 values in Table V are computed based on the likelihood ratio of models with all five metrics and the null model. They show the cumulative contribution of the five metrics. We observe from p-values and values of R^2 in Table III, that our metrics TTFF, FAR, and OFR display the ability to correctly predict the number of post-release defects. In the following, we discuss these results in more detail.

The prediction of the number of post-release defects within six months. From all the analysed metrics, including MTBF and AVT, our OFR metric is the only metric with significant coefficients (see Table III). The marginal McFadden's R^2 of OFR is up to 0.13, the highest value among the metrics. Therefore OFR has the ability to predict the number of six months post-release defects. As shown in

Table III
NEGATIVE BINOMIAL RESULTS FOR 5, 10, 15, 20, 25 AND 30 DAYS TESTING PERIOD (SIGNIFICANT p-VALUES AND R^2 ARE HIGHLIGHTED IN BOLD).

	Six month post-release defects					One year post-release defects					Two years post-release defects				
	Estimate	Std. Error	z value	R^2	Pr(>|z|)	Estimate	Std. Error	z value	R^2	Pr(>|z|)	Estimate	Std. Error	z value	R^2	Pr(>|z|)
Results of a 5 days testing period															
TTFF	0.91	1.90	0.48	0.00	0.63	2.86	1.46	1.96	**0.02**	**0.04**	0.47	1.47	0.32	0.00	0.74
FAR	-0.32	1.52	-0.21	0.00	0.83	-3.48	1.08	-3.21	**0.10**	**0.00**	-0.57	1.15	-0.49	0.00	0.62
OFR	-3.81	5.57	-0.68	**0.01**	0.49	-11.88	5.76	-2.06	**0.05**	**0.03**	-6.14	4.84	-1.27	**0.03**	0.20
MTBF	1.70	1.89	0.90	**0.01**	0.36	-2.75	1.53	-1.80	**0.04**	0.07	1.27	1.48	0.86	0.01	0.38
Results of a 10 days testing period															
TTFF	0.44	1.22	0.37	0.00	0.71	0.69	1.08	0.64	0.00	0.52	0.69	0.99	0.69	**0.01**	0.48
FAR	-1.11	1.88	-0.59	0.00	0.55	-2.32	1.74	-1.33	**0.02**	0.18	-1.57	1.51	-1.04	**0.02**	0.29
OFR	-12.75	7.13	-1.79	**0.07**	0.07	-14.76	7.31	-2.02	**0.05**	**0.04**	-13.22	5.60	-2.36	**0.10**	**0.01**
MTBF	0.18	0.79	0.23	0.00	0.81	0.29	0.75	0.39	0.00	0.69	0.45	0.63	0.72	**0.01**	0.47
Results of a 15 days testing period															
TTFF	-0.11	0.72	-0.16	0.00	0.87	1.38	0.67	2.04	**0.06**	**0.04**	0.09	0.59	0.16	0.00	0.87
FAR	-0.23	1.77	-0.14	0.00	0.89	-4.21	1.66	-2.52	**0.08**	**0.01**	-0.82	1.45	-0.56	0.00	0.57
OFR	-13.45	9.62	-1.40	**0.04**	0.16	-28.33	9.28	-3.05	**0.12**	**0.00**	-16.20	7.74	-2.09	**0.08**	**0.03**
MTBF	0.42	1.09	0.39	0.00	0.69	-1.45	0.95	-1.53	**0.03**	0.12	0.53	0.89	0.59	0.00	0.55
Results of a 20 days testing period															
TTFF	0.07	0.22	0.32	0.00	0.74	0.29	0.20	1.47	**0.03**	0.14	0.22	0.16	1.36	**0.04**	0.17
FAR	-0.85	0.81	-1.05	**0.03**	0.29	-2.16	1.03	-2.10	**0.06**	**0.03**	-1.36	0.71	-1.91	**0.08**	**0.05**
OFR	-18.25	7.80	-2.34	**0.13**	**0.01**	-21.72	8.41	-2.58	**0.08**	**0.00**	-22.55	6.45	-3.49	**0.22**	**0.00**
MTBF	0.01	0.26	0.02	0.00	0.98	-0.28	0.24	-1.18	**0.02**	0.23	0.21	0.17	1.26	**0.03**	0.20
Results of a 25 days testing period															
TTFF	0.04	0.16	0.27	0.00	0.78	0.24	0.13	1.80	**0.04**	0.07	0.05	0.11	0.53	0.00	0.59
FAR	-1.31	1.06	-1.23	**0.03**	0.21	-3.18	1.08	-2.93	**0.10**	**0.00**	-1.14	0.94	-1.22	**0.03**	0.22
OFR	-21.57	9.78	-2.21	**0.12**	**0.02**	-30.81	9.35	-3.29	**0.12**	**0.00**	-19.75	8.54	-2.31	**0.11**	**0.02**
MTBF	-0.18	0.62	-0.29	0.00	0.77	-1.01	0.50	-2.03	**0.06**	**0.04**	0.51	0.41	1.24	**0.03**	0.21
Results of a 30 days testing period															
TTFF	0.05	0.14	0.42	0.00	0.67	0.20	0.12	1.63	**0.04**	0.10	0.05	0.09	0.58	0.00	0.56
FAR	-1.41	1.08	-1.30	**0.04**	0.19	-3.19	1.14	-2.78	**0.10**	**0.00**	-1.18	0.95	-1.24	**0.03**	0.21
OFR	-24.86	11.33	-2.19	**0.12**	**0.02**	-34.58	10.99	-3.15	**0.12**	**0.00**	-22.36	9.77	-2.29	**0.11**	**0.02**
AVT	-0.00	0.03	-0.04	0.00	0.96	0.05	0.02	2.22	**0.06**	**0.02**	0.01	0.02	0.33	0.00	0.73
MTBF	-0.18	0.62	-0.30	0.00	0.76	-0.97	0.51	-1.89	**0.05**	0.06	0.53	0.41	1.29	**0.03**	0.19

Table V, the average precision of the predictions is 86%. Moreover, OFR is able to perform these predictions within just 20 days of usage across the 18 analyzed versions.

The prediction of the number of post-release defects within one year. All our metrics, TTFF, FAR, and OFR have significant coefficients (see Table III). They are able to predict the number of one year post-release defects within just 5 days of usage. In contrast, the MTBF metric requires at least 25 days to produce a significant relation with the number of one year post-release defects, and similarly, AVT requires 30 days. The average precision of these predictions is 66%, as shown in Table V. Moreover, our TTFF metric produces the best values when computed early in the application usage period.

The prediction of the number of post-release defects within two years. From all the analysed metrics, only our metrics, FAR, and OFR display the ability to predict the number of two years post-release defects (see Table III). The marginal McFadden's R^2 of OFR is up to 0.22, showing a good predictive power. OFR is able to perform these predictions in as little as 10 days of usage. As listed in Table V, the average precision of the predictions is 73%. Our FAR metric produces its best values when computed for a 20 days testing period.

In conclusion, TTFF, FAR, and OFR display strong abilities to predict the number of one year post-release defects, within just 5 days of usage. As listed in Table III, the predictive power of FAR and OFR increases as additional data is available. In contrast, the predictive power of TTFF sometimes deteriorates as additional data is available. In general, OFR metric outperforms all other metrics across all time periods and is robust against noises, showing consistent improvement as additional data is available.

Overall, as listed in Table V, the models are significant, show high R^2, and achieve an average precision of 86% in predicting the number of six months post-release defects, 66% for the number of one year post-release defects and 73% for the number of two years post-release defects.

RQ3: Are TTFF, FAR, and OFR more important than existing metrics AVT and MTBF for predicting the number of post-release defects?

To answer this research question, we analyse the results achieved by our multivariate models in the entire field testing period of our application *i.e.*, 30 days. These results are presented in Table III.

Although the correlations between the metrics are weak, they do exist. Under such conditions, Hayashi [12] noted that a univariate estimator would be inconsistent and error-prone. Therefore, we perform a multivariate analysis.

We observe that the FAR and OFR metrics are more important to predict the number of post-release defects than the traditional MTBF and AVT metrics. Specifically, FAR and OFR are the most important metrics for the prediction of the number of one year post-release defects. FAR and OFR have higher marginal R^2 values. Overall, OFR is the most important metric, as we noted before. It shows a stronger relation to the number of post-release defects across all time periods. Although TTFF appears not to be a good predictor for the number of post-release defects over a 30 days period, our previous research question (**RQ2**) has shown its ability to perform good predictions of the number of one year post-release defects when computed early in the testing period,

e.g., within the first 5 days. Therefore, managers and quality assurance personnel can make use of the three metrics to predict the number of post-release defects of an application to be released and better plan testing activities and the allocation of support resources. They can also decide to shorten the testing period based on the results of the metrics.

Table IV shows that the average usage time (AVT) can be successfully used to predict the number of one year post-release defects. However, this average usage time shows poor ability to predict the number of two years post-release defects. Because applications are generally used for much longer periods of time than the field testing period, it is infeasible to use AVT to predict the number of post-release defects of an application. MTBF displays a weak ability to predict the number of one year post-release defects, with a p-value that is almost significant (*i.e.*, 0.06), and a marginal R^2 of 0.05.

In conclusion, TTFF, FAR, and OFR, show higher abilities to predict the number of post-release defects compared to the traditional MTBF, and AVT metrics. Although AVT and MTBF are able to predict the number of one year post-release defects (to a lesser degree), they require at least a 25 days testing period to make these predictions. In contrast, it takes just 5 days of usage for TTFF, FAR, and OFR to predict the number of one year post-release defects. Moreover, AVT and MTBF are unable to predict the numbers of six months and two years post-release defects.

E. Threats to Validity

We now discuss the threats to validity of our study, following the guidelines for case study research [13].

Construct validity threats concern the relation between theory and observation. In this study, construct validity threats are mainly due to measurement errors. Each version of our enterprise software application has at least 50 users. However, users tend to fluctuate significantly between versions. Users may have different preferences that cause some users to behave differently from others. Some users are more casual and seldom use the application, while others use the application on a daily basis. Therefore, it may be more accurate if each version has roughly the same user base to ensure unbiased usage. Given the length of the testing period and the large number of studied versions, we are not able to ensure such consistency in a real data set. An alternative would be to conduct controlled user studies to determine if this bias impacts the validity of our results. We can also split up the data based on the distribution of different usage patterns (*e.g.*, casual, medium and high usage). However, users join the field testing period at different times. We assume that all the users start the field testing at the same time. We adjust the usage times for all users by adopting a universal starting Day 1 for everyone. The universal day is incremented with every new day of usage by the user. The choice affects the length of testing periods (*e.g.*, it may take longer than 5 days to have a good number of users with a complete 5 days of usage of the application). However, we observe that a large majority of users enrol very early at the beginning of the testing period.

Threats to *internal validity* do not affect this study, being an exploratory study [13]. We do not claim causation, but simply show relations between our metrics and the number of post-release defects. Using our metrics and knowledge about the required resources for prior releases, managers and quality assurance personnel can better plan and organize their limited testing and support resources. They can also reduce the length of the testing period.

Conclusion validity threats concern the relation between the treatment and the outcome. We pay attention not to violate assumptions of the performed statistical tests. Moreover, we mainly use non-parametric tests that do not require assumptions about the distribution of data set.

Threats to *external validity* concern the possibility to generalize our results. We have validated our metrics on 18 versions of an enterprise mobile software application. This represents a large number of industrial software systems. Gaining access to such industrial data is hard and open source systems do not provide similar data from field testing. Nevertheless, further validation on a larger set of software applications is desirable, considering applications from different domains, as well as several applications from the same domain.

IV. Related work

We now discuss work on the reliability of software systems and defect predictions.

A. Software Reliability Improvement

The IEEE Standard Computer Dictionary [14] defines the reliability of a software system to be the ability to perform the required functions under stated conditions for a specified period of time. The reliability is often measured by the failure rate of the software, which is an important factor to determine user satisfaction. Software reliability can be improved by mining field testing data to assess the quality of the software.

Gonzalez *et al.* [15] condense the vast amounts of field information into metric scores to evaluate usability of mobile devices using data mining techniques. Sarbu *et al.* [16] create metrics to measure the runtime behavior of Windows device drivers using I/O traffic as input. The study aimed to improve the quality of products by improving the testing scenarios for the drivers through a pinpoint of hotspots in execution profiles. Mockus [17] assesses system reliability using information readily available in most systems, such as alarm information and the number of defect fixes issued. Mockus defines reliability as the amalgamation of three aspects: the total runtime, the number of outages and the duration of outages. Keene [3] proposes a model to predict reliability based upon characteristics of the system and the capability of the developing organization. The model transforms the latent defect density into an exponential reliability growth curve over time, as failures surface and the underlying defects

are found and removed. Buckley *et al.* [18] argue that user satisfaction and service delivery directly affect each other. They measure service variables and customer attributes to show that the return user satisfaction (and thereby investment) is exponential to the amount of effort spent fixing defects. Similar to the aforementioned approaches, we mine the field testing data to evaluate the reliability of software. Moreover, we predict the number of defects that occur after an application is released using as little as 5 days of field testing data.

B. Defect Prediction

Several studies focus on predicting the location of the faults using software development repositories. For example, Kim [19], [20] discusses a method called bug caching, initially proposed by Hassan and Holt [21], which records the historic locations of bugs in order to predict where future bugs will occur with the highest certainty. This technique could be used to isolate reported bugs in post-release based on pre-release data. Joshi *et al.* [22] use a similar approach for bug prediction by analyzing the recent locality of bugs. They use the recorded history of the Eclipse project in order to predict the occurrence of future bugs.

Some studies propose the use of source code metrics for bugs prediction. For example, Khoshgoftaar *et al.* [23] report good results from a combination of code metrics and knowledge from problem reporting databases for bug predictions. Moser *et al.* [24] however show that process metrics outperform source code metrics as predictors of future bugs. Hangal *et al.* [25] introduce a tool called DIDUCE which formulates invariants obeyed by a program. The tool predicts and isolates bugs based on the number of invariants violated by the program. Different from aforementioned approaches, which use software development repositories, our study analyzes the reliability of software to predict the number of defects after releases using pre-release field testing data.

Other researchers focus on using temporal information for bug prediction. Bernstein *et al.* [26] use temporal aspects of data (*i.e.*, the number of revisions and corrections recorded in a given amount of time) to predict the location of defects. The results can predict whether a source file has a defect with 99% accuracy. Arisholm and Briand [27] propose the use of code quality, class structure, changes in class structure, and the history of class-level changes and bugs to predict defects in classes. They perform a cost-effectiveness analysis and show that the estimated potential savings in verification effort of their model is about 29%. Graves *et al.* [28] develop several statistical models to evaluate characteristics of a module's change history likely to be good indicators of future bugs. They conclude that the sum of contributions from all the changes to a module in the history is a good predictor of bugs. Nagappan and Ball [29] show that relative code churn metrics are good predictors of bug density in systems. Askari and Holt [30] provide a list of mathematical models to predict where the next bugs are likely to occur. Yu *et al.* [31] propose a mathematical model to estimate the total number of remaining bugs in a system. The models would be an ideal technique to constantly monitor the files that are under the threat of incurring bugs. When our metrics predict a high number of post-release defects for an application, models from Askari and Holt [30] can be used to determine the location of the defects using pre-release data.

V. Conclusion and Future work

Mining software repositories is rapidly emerging as an important area of software engineering research with strong and positive implications on software practices. The software engineering literature is rich with techniques to predict post-release defects using development repositories. In this paper, we propose the use of another rarely used yet readily available repository, the pre-release problem reports repository that is created during the pre-release field testing (*i.e.*, beta testing) of an application. We propose three metrics to capture various aspects of the reliability of an application based on its usage during beta testing. We compare the performance of the metrics against two often-used metrics AVT and MTBF.

Using data from 18 versions of an enterprise software application, we found that our metrics complement AVT and MTBF in describing the reliability of a software system. Moreover, we also found that our proposed metrics are more important to predict the number of post-release defects than these often-used metrics, while in the same time being able to produce good predictions within a much shorter time frame, *i.e.*, within just 5 days of a pre-release testing period.

In future work, we plan to conduct user studies to better understand the ability of our proposed metrics in predicting future defects. We also plan to explore the benefits of combining our metrics with development metrics (*i.e.*, source code metrics and process metrics) to predict defects.

Acknowledgements. We would like to thank Dr. Ahmed E. Hassan at Queen's University for his suggestions on the analysis process and construction of the metrics. We would also like to thank Dr. Bram Adams at Queen's University, for his feedback on our work.

We are grateful to Research In Motion (RIM) for providing access to the data used in this study. The findings and opinions expressed in this paper are those of the authors and do not necessarily represent or reflect those of RIM and/or its subsidiaries and affiliates. Moreover, our results do not in any way reflect the quality of RIM's products.

References

[1] J. Musa, A. Iannino, and K. Okumoto, *Software Reliability. Measurement, Prediction, Application*. McGraw-Hill, 1987.

[2] A. Mockus and D. Weiss, "Interval quality: relating customer-perceived quality to process quality," in *ICSE '08: Proceedings of the 30th international conference on Software engineering*. NY, USA: ACM, 2008, pp. 723–732.

[3] S. Keene, "Modeling software R&M characteristics," *ASQC Reliability Review, Part I and II*, vol. 17, no. 2&3, pp. 13–22, June 1997.

[4] N. Bettenburg, R. Premraj, T. Zimmermann, and S. Kim, "Duplicate bug reports considered harmful...really?" sep. 2008, pp. 337 –345.

[5] A. Marcus, D. Poshyvanyk, and R. Ferenc, "Using the conceptual cohesion of classes for fault prediction in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 34, no. 2, pp. 287–300, 2008.

[6] H. S. Chae, Y. R. Kwon, and D.-H. Bae, "A cohesion measure for object-oriented classes," *Softw. Pract. Exper.*, vol. 30, pp. 1405–1431, October 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=362441.362449

[7] M. Hollander and D. A. Wolfe, *Nonparametric Statistical Methods*, 2nd ed. John Wiley and Sons, inc., 1999.

[8] P. McCullagh and J. Nelder, *Generalized Linear Models*, 2nd ed. Chapman and Hall, 1989.

[9] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340–355, 2005.

[10] D. McFadden, "Conditional logit analysis of qualitative choice behavior," *Frontiers in Econometrics*, 1974.

[11] R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*. Addison-Wesley, 1999.

[12] F. Hayashi, *Econometrics*. Princeton University Press, 2000.

[13] R. K. Yin, *Case Study Research: Design and Methods - Third Edition*, 3rd ed. SAGE Publications, 2002.

[14] "IEEE Standard computer dictionary. A Compilation of IEEE Standard computer glossaries," *IEEE Std 610*, p. 1, 1991.

[15] M. P. González, J. Lorés, and A. Granollers, "Enhancing usability testing through datamining techniques: A novel approach to detecting usability problem patterns for a context of use," *Information and Software Technology*, vol. 50, no. 6, pp. 547–568, 2008.

[16] C. Sârbu, A. Johansson, N. Suri, and N. Nagappan, "Profiling the operational behavior of os device drivers," in *ISSRE '08: Proceedings of the 2008 19th International Symposium on Software Reliability Engineering*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 127–136.

[17] A. Mockus, "Empirical estimates of software availability of deployed systems," in *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. NY, USA: ACM, 2006, pp. 222–231.

[18] M. Buckley and R. Chillarege, "Discovering relationships between service and customer satisfaction," in *ICSM '95: Proceedings of the International Conference on Software Maintenance*. Washington, DC, USA: IEEE Computer Society, 1995, p. 192.

[19] S. Kim, "Adaptive bug prediction by analyzing project history," Ph.D. dissertation, Santa Cruz, CA, USA, 2006, adviser-Whitehead,Jr., E. James.

[20] S. Kim, T. Zimmermann, E. J. Whitehead Jr., and A. Zeller, "Predicting faults from cached history," in *ICSE '07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 489–498.

[21] A. E. Hassan and R. C. Holt, "The top ten list: Dynamic fault prediction," in *ICSM '05: Proceedings of the 21st IEEE International Conference on Software Maintenance*. DC, USA: IEEE Computer Society, 2005, pp. 263–272.

[22] H. Joshi, C. Zhang, S. Ramaswamy, and C. Bayrak, "Local and global recency weighting approach to bug prediction," in *MSR '07: Proceedings of the Fourth International Workshop on Mining Software Repositories*. Washington, DC, USA: IEEE Computer Society, 2007, p. 33.

[23] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Data mining for predictors of software quality," *International Journal of Software Engineering and Knowledge Engineering*, vol. 9, no. 5, 1999.

[24] R. Moser, W. Pedrycz, and G. Succi, "A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction," in *Proceedings of the International Conference on Software Engineering*. New York, NY, USA: ACM, 2008, pp. 181–190.

[25] S. Hangal and M. S. Lam, "Tracking down software bugs using automatic anomaly detection," in *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, NY, USA: ACM, 2002, pp. 291–301.

[26] A. Bernstein, J. Ekanayake, and M. Pinzger, "Improving defect prediction using temporal features and non linear models," in *IWPSE '07: Ninth international workshop on Principles of software evolution*. NY, USA: ACM, 2007, pp. 11–18.

[27] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," in *ISESE '06: Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*. NY, USA: ACM, 2006, pp. 8–17.

[28] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy, "Predicting fault incidence using software change history," *IEEE Transactions on Software Engineering*, vol. 26, no. 7, pp. 653–661, 2000.

[29] N. Nagappan and T. Ball, "Use of relative code churn measures to predict system defect density," in *ICSE '05: Proceedings of the 27th international conference on Software engineering*. NY, USA: ACM, 2005, pp. 284–292.

[30] M. Askari and R. Holt, "Information theoretic evaluation of change prediction models for large-scale software," in *MSR '06: Proceedings of the 2006 international workshop on Mining software repositories*. NY, USA: ACM, 2006, pp. 126–132.

[31] T.-J. Yu, V. Y. Shen, and H. E. Dunsmore, "An analysis of several software defect models," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1261–1270, 1988.