# Using Historical In-Process and Product Metrics for Early Estimation of Software Failures

Nachiappan Nagappan, Thomas Ball, Brendan Murphy
*Microsoft Research*
*{nachin, tball, bmurphy}@microsoft.com*

## Abstract

*The benefits that a software organization obtains from estimates of product quality are dependent upon how early in the product cycle that these estimates are available. Early estimation of software quality can help organizations make informed decisions about corrective actions. To provide such early estimates we present an empirical case study of two large scale commercial operating systems, Windows XP and Windows Server 2003. In particular, we leverage various historical in-process and product metrics from Windows XP binaries to create statistical predictors to estimate the post-release failures/failure-proneness of Windows Server 2003 binaries. These models estimate the failures and failure-proneness of Windows Server 2003 binaries at statistically significant levels. Our study is unique in showing that historical predictors for a software product line can be useful, even at the very large scale of the Windows operating system.*

## 1. Introduction

The benefits software organizations obtain from estimates of product quality are dependent upon how early in the product cycle these estimates are available. When product quality information is available late in the software development cycle, corrective actions tend to be expensive [4]. We address this issue by using in-process and product metrics to make estimates of post-release failures early in the software development cycle, during the implementation and testing phases. Such estimates can help focus testing, code and design reviews and affordably guide corrective actions.

Many different internal and external metrics are related to product quality. An internal metric (measure), such as cylomatic complexity [26], is a measure derived from the product itself [19]. An external measure of a product is derived from the external assessment of the behavior of the system [19]. For example, the number of failures found in test is an external measure. The ISO/IEC standard [16] states that an internal metric is of little value unless there is evidence that it is related to an externally-visible attribute. Internal metrics have been shown to be useful as early indicators of externally-visible product quality when they are related in a statistically significant and stable way to the field quality/reliability of the product. The validation of internal metrics requires a convincing demonstration that (1) the metric measures what it purports to measure and (2) the metric is associated with an important external metric, such as field reliability, maintainability, or fault-proneness [11].

Our primary research motivation is to investigate the utility of using in-process and product metrics to estimate post-release failures/failure-proneness. Failure-proneness is the probability that a particular software element (such as a binary) will fail in operation in the field. The higher the failure-proneness of the software, the lower the reliability and the quality of the software produced. While software fault-proneness can be measured before deployment (i.e. the count of faults per structural unit such as faults per line of code) it is not necessarily an indication of failure-proneness. Conversely, by definition, failure-proneness cannot be directly measured before deployment [12]. Identifying associations among software attributes and the system failure-proneness in prior releases allows us to estimate the failure-proneness of software under development. Table 1 more formally states our research hypotheses.

**Table 1: Research Hypotheses**

| | **Hypotheses** |
|---|---|
| $H_1$ | Historical in-process and product metrics from Windows XP-SP1 can be used to estimate the post-release failures for Windows Server 2003 at statistically significant levels. |
| $H_2$ | Historical in-process and product metrics from Windows XP-SP1 can be used to identify failure-prone binaries for Windows Server 2003 at statistically significant levels. |
| $H_3$ | Hypothesis $H_1$ and $H_2$ also hold for individual components in Windows Server 2003 |

This paper is organized as follows. Section 2 reviews the related work. Section 3 describes metrics collection of the in-process and product metrics. Section 4 presents our empirical case study. Section 5 repeats our study at a finer-level of granularity to show that the basic hypotheses hold at various levels of scale in the software under consideration. Section 6 identifies the threats to validity and Section 7 presents our conclusions and future work.

## 2. Related Work

We now discuss some of the earlier work related to investigations of complexity metrics and quality, code churn, and using historical metrics as predictors of software quality in large software systems.

Structural object-orientation (O-O) measurements, such as those in the CK O-O metric suite [8], have been used to evaluate and predict fault-proneness [2, 5, 6]. Research on fault-proneness has focused on two areas: (1) the definition of metrics to capture software complexity and testing thoroughness and (2) the identification of and experimentation with models that relate software metrics to fault-proneness [9]. These metrics can be an useful early internal indicator of externally-visible product quality [2, 29, 30].

Khoshgoftaar et al.[18] studied two consecutive releases of a large legacy system for telecommunications. The system contained over 38,000 procedures in 171 modules. Discriminant analysis identified fault-prone modules based on 16 static software product metrics. Their model when used on the second release showed a type I and II misclassification rate of 21.7%, 19.1% respectively and an overall misclassification rate of 21.0%. Case studies performed using classification trees [19] with statement, control-flow, process and execution metrics on multiple releases of a successive large scale legacy telecommunications system written in Protel (a procedural language) indicates the efficacy of classification trees towards identification of fault-prone components.

Ostrand et al. [28] use information of file status such as new, changed, unchanged files along with other explanatory variables (such as lines of code, age, prior faults etc.) as predictors in a negative binomial regression equation to predict the number of faults in a multiple release software system (Size of the last release was 538 KLOC). The predictions made using binomial regression model were of a high accuracy for faults found in both early and later stages of development [28]. Denaro et al. [10] calculated 38 different software metrics (lines of code, halstead software metrics, nesting levels, cyclomatic complexity, knots, number of comparison operators, loops etc.) for the open source Apache 1.3 and Apache 2.0 projects. Using PCA, they selected a subset of nine of these metrics which explained 95% of the total data variance. Using combinations of these nine metrics, logistic regression models were built using the data from the Apache 1.3 project and verified against the Apache 2.0 project.

Zimmermann et al. [31] performed one of the largest studies on mining historical data. They mined 8 large scale open source systems (IBM Eclipse, Postgres, Koffice, gcc, Gimp, JBOSS, Jedit and Python). They predict where future changes will take place in systems. The top three recommendations made by their system contained a correct location for future change with an probability of 70%. Mockus et al. [22] predict the customer perceived quality using logistic regression for a commercial telecommunications system (of size seven million lines of code) by utilizing external factors like hardware configurations, software platforms, amount of usage and deployment issues. They observed an increase in probability of failure by twenty times by accounting for such measures in their prediction equations.

While this paper shares similar goals with the above work, our case study employs a different suite of metrics that integrate process and product metrics including code coverage and static source code metrics. Our case study also is much larger in terms of scale, as it represents a widely used commercial operating system. Previous studies have been performed on telecommunications software (with a distinct user profile) and open source software systems. We also use post-release failures as they are of critical and economic importance to organizations.

This work is part of the MetriZone project at Microsoft Research, whose goal is to make early estimates of software quality to predict post-release failures. In our prior studies [24] we investigated the use of a set of relative code churn measures in isolation as predictors of software defect density. The relative churn measures are normalized values of the various measures obtained during the churn process. We have also used defects found by static analysis tools as early indicators of pre-release defect density [23] and code complexity metrics [25] as early indicators of software failures. Our previous studies used random splitting in our prediction fits. Two thirds of our sample data points were used to predict the failures in the remaining one third [9]. Our current study uses two different operating system versions and a different suite of metrics to build our prediction sets.

## 3. Metrics Collection

The data for our experiment is collected from the Windows XP-SP1 and Windows 2003 Server operating systems. Table 2 shows a few simple measures in terms of the size of the two systems used in our case study. All the data collected is at a binary level of granularity. We measure failures as defined by the IEEE standard [1] as the inability of a system or component to perform its required functions within specified performance requirements. To clarify between fault, defect and failure, when software is being developed, a person makes an error that results in a physical fault (or defect) in a software element. When this element is executed, traversal of the fault/defect may put the element (or system) into an erroneous state. When this erroneous state results in an externally visible anomaly, we say that a failure has occurred [1].

**Table 2: Analyzed case study data**

| | **Windows XP – SP1** | **Windows Server 2003** |
|---|---|---|
| Domain of software | Operating system – Desktop market | Operating system – Server market |
| Size (LOC) | 1.58 M LOC | 29.28 M LOC |
| Binaries | 569 | 2075 |
| Team Size | Range of (‘000’s) | Range of (‘000’s) |
| Post-release failures | Collected for six months | Collected for six months |
| Product and Process metrics collection | Automated | Automated |

Figure 1 shows the overall timeline in which the product and process metrics were collected for Windows XP-SP1 and Windows Server 2003. The failures are collected for a time of six months after the release of the software to the field. The data is collected by synchronizing with the source control systems that allows us to identify the fixes for failures occurring in the field. We measure only the failures that have an associated change of the source code.

### 3.1 Process Metrics

Process metrics evolve over time. They are most useful when measured over a period of time and compared to a baseline measurement. We use Windows XP to form the baseline for Windows XP-SP1 and Windows 2000 to form the baseline for Windows Server 2003. The collection of the process metrics is completely automated via a version control system. The three process measures we measure over time are:

**Delta Lines** measures the number of lines of code that have “changed” between two versions (this includes added, deleted and modified lines, as computed by the textual diff algorithms of the version control system). The delta lines of code are the sum of the added, deleted and modified lines between the two versions. In our context we compute the delta lines of code that changed between the final versions of Windows XP and Windows XP-SP1. The delta lines of code is computed using the source lines of code and rolled up to the level of binaries.

**Frequency of Churn** is defined as the number of times a binary is worked upon (check-ins) during the development process. Unstable binaries often churn repeatedly due to multiple fixes thereby causing a “tail of defects” to be observed.

**The number of developers** who contributed code to a binary is measured between the baseline and new version. This measure is used to quantify the overall development/team organization complexity. Larger teams typically have more issues around coordination, communication, and understanding program dependences. The size and organization of the teams can affect product quality to a large degree [7].

### 3.2 Product Metrics

Product metrics are derived from the system itself. The metrics (coverage and complexity) are collected at a specific time instance, in our case just before the external release of the system. The coverage metrics (block and arc) quantify the testing effort performed on the binaries. The code complexity metrics quantify the relationship between complex code and failures. The individual description of the metrics is presented below.

**Block coverage:** A (basic) block is a set of contiguous instructions (code) in the physical layout of a binary that has exactly one entry point and one exit point. Calls, jumps, and branches mark the end of a block. A block typically consists of multiple machine-code instructions. The number of blocks covered during testing constitutes the block coverage measure.

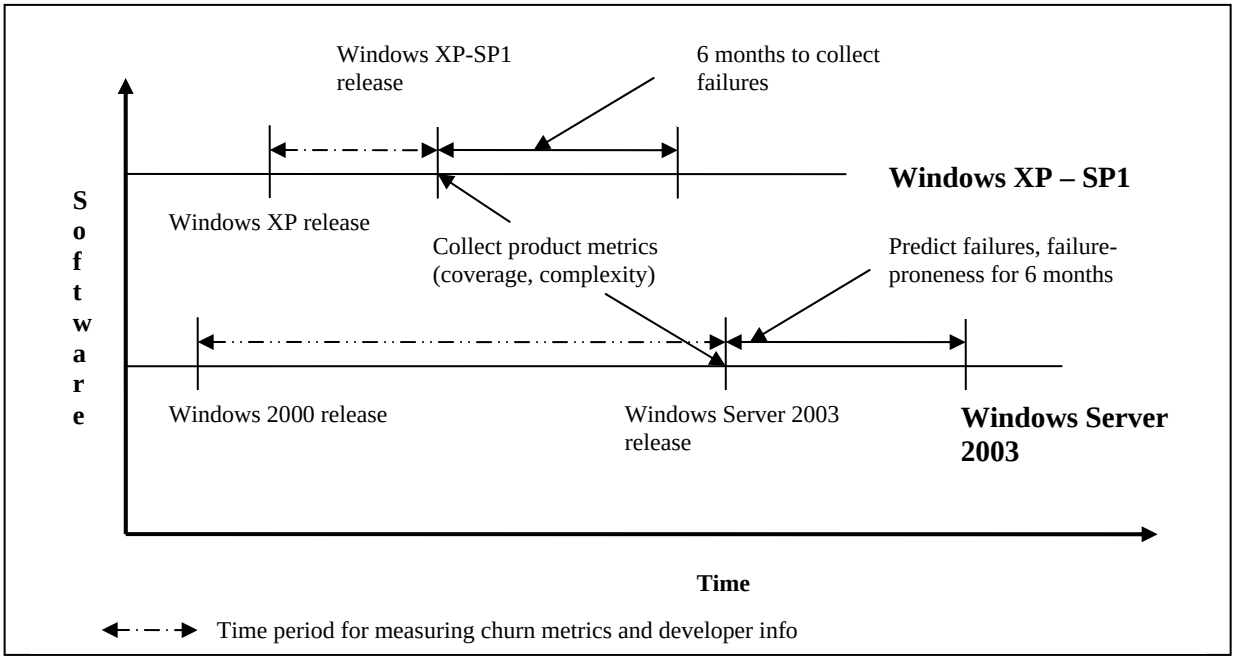


**Figure 1: Metrics collection timeline**

**Arc coverage:** Arcs between blocks represent the transfer of control between basic blocks (due to conditional and unconditional jumps, as well as control falling through from one block to another). Similar to block coverage the proportion of arcs covered in a binary constitute the arc coverage.

**Code complexity:** McCabe's Cyclomatic Complexity metric [21] measures the number of linearly-independent paths through a program module. It is used to measure code complexity in a binary.

## 4. Empirical Case Study

We now discuss the case study we performed using the historical in-process and product metrics on the Windows operating systems. In order to demonstrate the utility of the statistical model to predict failures/failure-proneness we use two approaches. We use logistic regression to estimate failure-proneness and a multiple regression approach to estimate the absolute values of the post-release failures.

Sections 4.1 and 4.2 discuss the estimation of the absolute values of the failures and failure-proneness for all the 2075 binaries from the Windows Server 2003 dataset (each binary serves as a data point). We also discuss the comparison between the actual and the estimated values for all the 2075 Windows Server 2003 binaries. One difficulty associated in general with regression techniques is multicollinearity among the metrics, which can lead to inflated variance in the prediction of post-release failures/failure-proneness. Multicollinearity occurs when the predictors are strongly correlated. Table 3 shows the correlation between the in-process and product metrics. We also observe strong correlations between the delta lines of code, frequency of churn and number of developers indicating that binaries that have a larger churn size undergo more edits by a large number of developers.

To avoid problems due to multicollinearity we use Principal Component Analysis (PCA) [17]. With PCA, a small number of uncorrelated linear combinations of metrics (that account for as much sample variance as possible) are selected for use in regression (linear or logistic). A logistic regression equation [13] or multiple regression equation can be built to model data using the principal components as the independent variables. We performed PCA on the process and product metrics for XP-SP1. Table 4 shows the individual and cumulative variances for each principal component and their respective eigen values for the XP-SP1. The eigen values represent the relative importance of each principal component in explaining the variance in the initial dataset.

**Table 4: Windows XP-SP1 PCA information**

| Principal Component | Extraction Sums of Squared Loadings | | |
|---|---|---|---|
| | Eigen value | % of Variance | Cumulative % |
| 1 | 2.970 | 49.508 | 49.508 |
| 2 | 1.817 | 30.279 | 79.786 |
| 3 | .597 | 9.956 | 89.743 |
| 4 | .427 | 7.111 | 96.854 |

**Table 3: Correlation matrix between in-process and product metrics**

| | | Delta lines of code | Frequency of churn | Number of developers | Block Coverage | Arc Coverage | Code Complexity |
|---|---|---|---|---|---|---|---|
| Delta lines of code | Correlation Coefficient | 1.000 | .825 | .569 | .240 | .236 | .267 |
| | Sig. (2-tailed) | . | (<.0005) | (<.0005) | (<.0005) | (<.0005) | (<.0005) |
| Frequency of churn | Correlation Coefficient | | 1.000 | .684 | .228 | .228 | .326 |
| | Sig. (2-tailed) | | . | (<.0005) | (<.0005) | (<.0005) | (<.0005) |
| Number of developers | Correlation Coefficient | | | 1.000 | .199 | .212 | .252 |
| | Sig. (2-tailed) | | | . | (<.0005) | (<.0005) | (<.0005) |
| Block Coverage | Correlation Coefficient | | | | 1.000 | .994 | -.084 |
| | Sig. (2-tailed) | | | | . | (<.0005) | (.045) |
| Arc Coverage | Correlation Coefficient | | | | | 1.000 | -.058 |
| | Sig. (2-tailed) | | | | | . | (.170) |
| Code Complexity | Correlation Coefficient | | | | | | 1.000 |
| | Sig. (2-tailed) | | | | | | . |

We select a stopping rule for selection of principal components that account for greater than 95% of the total cumulative variance [10] for use in the building of our statistical models. Among the extracted components the in-process and product measures that dominate in principal component one were the frequency of churn, number of developers and the delta lines of code. For principal component two, the arc and block coverage measures were the strongest contributors and for principal component three, the code complexity dominated.

## 4.1 Failure Estimation

We utilize a multiple regression technique (by analyzing the XP-SP1 metrics and failure data) to predict the number of post-release failures that will occur for a period of six months after the release of Windows Server 2003. We then compare the estimated values of the post-release failures with the actual values obtained from the field. The multiple regression equation is of the general form shown in Equation 1,

$$Y = c + a_1PC1 + a_2PC2 + \ldots + a_nPCn, \quad (1)$$

where $a_1$, $a_2$ …., $a_n$ are regression coefficients and PC1, PC2 …., PCn are the principal components (from the analysis of the XP-SP1 metrics) and Y is the independent variable.

Multiple regression using the 4 principal components for the XP-SP1 data yields an $R^2$ value of 0.450 ($F = 115.183$, $p<0.0005$). The $R^2$ value is the ratio of the regression sum of squares to the total sum of squares. As a ratio, it takes values between 0 and 1, with larger values indicating more variability explained by the model and less unexplained variation. The F-ratio tests the null hypothesis that all regression coefficients are zero at statistically significant levels. The $R^2$ value and t-test values indicates that the regression model built is statistically significant.

Another measure associated with $R^2$ is the Adjusted $R^2$ (Adj-$R^2$) measure. Adj-$R^2$ explains for any bias in the $R^2$ measure by taking into account the degrees of freedom of the independent variables (i.e. the metrics) and the sample population (total dataset size). Equation 2 gives the computation of Adj-$R^2$.

$$\text{Adjusted } R^2 = \frac{R^2 - (V^i - 1)}{(n - V^i) * (1 - R^2)} \quad (2)$$

where n is the number of samples used to build the regression model and $V^i$ is the number of independent variables used to build the regression model. The Adj-$R^2$ for the XP-SP1 dataset is 0.446. The minor difference between the $R^2$ and Adj-$R^2$ values indicates that there is no bias in our fit due to the independent variables (probably because of the large size of our dataset). Throughout the rest of the paper we present

the Adj-$R^2$ in addition to the $R^2$ values to explain any bias caused due to the use of the independent variables.

Using this regression model, we predict the post-release failures for Windows Server 2003. In order to numerically quantify the sensitivity of prediction between the actual and estimated failures we run a Spearman rank correlation. The Spearman rank correlation is a commonly-used robust correlation technique [14] because it can be applied even when the association between elements is non-linear. Table 5 presents results of the Spearman correlation between the actual and estimated failures.

**Table 5: Correlation results between estimated failures and actual failures**

| | | **Estimated failures** |
|---|---|---|
| **Actual failures** | Spearman Correlation | 0.635 |
| | Sig. (2-tailed) | p < .0005 |

The correlations results in Table 5 are statistically significant and strongly indicate that with an increase in the estimated value of failures, there is an increase in the actual value. The strength of the correlations indicates the sensitivity of the predictions. These results indicate that *historical in-process and product metrics from Windows XP-SP1 can estimate the post-release failures at statistically significant levels for Windows Server 2003 ($H_1$).*

### 4.2 Failure-proneness Estimation

To identify failure prone binaries we employ a logistic regression technique. Failure-proneness is a probability measure (varying between 0 and 1). The higher the failure-proneness, the higher is the probability of experiencing a post-release failure. The general form of a logistic regression equation is given as in Equation 3:

$$\text{Probability (p)} = \frac{e^{(c+a1X1+a2X2+\ldots)}}{1 + e^{(c+a1X1+a2X2+\ldots)}} \quad (3)$$

where a1,a2 are the regression predicted constants and the X1,X2…are the independent variables used for building the logistic regression mode.

We use the XP-SP1 data to build our logistic regression model and evaluate the data from the Windows Server 2003 to determine the accuracy of the model. Based on the XP-SP1 data that was used to build the logistic regression equation we obtain the predicted probability of failure as shown in Equation 3. For the logistic regression models we compute the Nagelkerke's $R^2$ [26]. The Nagelkerke's $R^2$ is computed as shown in Equation 4,

$$\text{Nagelkerke's } R^2 = 1 - e^{(-LR/n)} \quad (4)$$

where LR is the likelihood ratio chi-square for the complete model and 'n' is the number of observations in the dataset. The Nagelkerke's $R^2$ = 0.342 for the model built using the principal components generated using the Windows XP-SP1 data.

A point to be noted is that the Nagelkerke's $R^2$ measure is not a true $R^2$ measure as in multiple regression and should not be compared with multiple regression. These $R^2$ values are presented in order to show the overall consistency in the fit of the logistic regression equation to explain the failure-prone (or not) binaries.

We compare the predicted probability for failure of a binary with the actual failures in the field for all the 2075 binaries binaries. In order to numerically quantify this relationship and evaluate the sensitivity of our failure-proneness prediction we run the Spearman correlation between the predicted probabilities of failures with the actual failures for Windows Server 2003 the results of which are shown in Table 6.

**Table 6: Correlation results between predicted probability (failure) and actual failures**

| | | **Estimated Prob (failure)** |
|---|---|---|
| **Actual failures** | Spearman Correlation | 0.631 |
| | Sig. (2-tailed) | P < .0005 |

Table 6 shows that the Spearman correlation coefficient is positive and statistically significant. These results show that our statistical model can help us flag failure-prone binaries early in the development process. In order to flag these binaries we can define, for example, binaries that have a predicted failure probability of 0.8 or greater to be failure-prone and then focus more testing, code and design reviews, fault-injection etc. on these binaries. Our results indicate that *historical in-process and product metrics from Windows XP-SP1 can be used to identify failure-prone modules for Windows Server 2003 at statistically significant levels ($H_2$).*

## 5 Component-Level Case Studies

Section 4 presented results in the aggregate, over all the binaries comprising XP-SP1 and Windows Server 2003. However, one expects that the components of a software system will have different distributions of failures. Thus, it makes sense to examine if the results observed in the large in Section 4 hold when we consider smaller components of the operating system. In order to investigate this scenario, we build

component-level models using data from Windows XP-SP1. In a highly simplified view, let us consider the organization of Windows as shown in Figure 2. All components that are shipped in Windows (XP/Server 2003) broadly belong to the three layers shown. The inner core consists of components that form the OS core like the kernel; the middleware consists of components that act as an interface with the kernel/core and the external layer consists of components that involve the user interface (shell). A component generally consists of many binaries.

We selected one component from each of these layers and built component-specific regression models using the XP-SP1 data for each of these components. These models are then used to predict the post-release failures and failure-proneness of the same component in Windows Server 2003. Let us call the three selected components (one from each of the architectural layers of Figure 2) A, B and C.

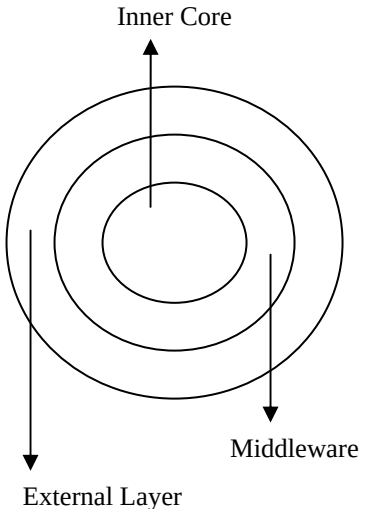


**Figure 2: Broad organization of Windows**

The components were selected based on the following two criteria: (1) the components are present in both Windows XP-SP1 and Windows Server 2003; (2) in Windows Server 2003 at least 25% of the binaries in the component should have churned compared to XP-SP1 so that we study different binaries and not the same ones as released previously.

## 5.1 Component A

Component A forms part of the inner core of the Windows operating system. The metrics extracted from Windows XP-SP1 for component A are obtained from 1697 files, which accounts for 936 KLOC. We use this data to generate principal components to avoid muticollinearity among the metrics, as before. Table 7 shows the generated principal components that account for a substantial cumulative variance (greater than 95%) of the data set. These three principal components are then input to the multiple regression and logistic regression equations. Amongst the three extracted principal components the measures that dominate in contribution towards principal component are, in order, the number of users, frequency of churn, code complexity and the delta lines of code. For principal component two, the major contributions are from the arc and block coverage.

**Table 7: Principal components for component A**

| Principal Component | Initial Eigen values | | |
|---|---|---|---|
| | Eigen value | % of Variance | Cumulative % |
| PC1 | 3.917 | 65.285 | 65.285 |
| PC2 | 1.588 | 26.462 | 91.747 |
| PC3 | .361 | 6.017 | 97.764 |

The $R^2$, Adj. $R^2$, and F-test indicate the strength and statistical significance of the built multiple regression model. The Nagelkerke $R^2$ are also presented for the logistic regression equations.

$R^2 = 0.783$, Adj-$R^2$=0.753, F=26.416, $p<0.0005$; Nagelkerke $R^2 = 0.362$

We use the corresponding metrics data for component A from Windows Server 2003 to compute estimates of post-release failures via the multiple regression equation and failure-proneness via the logistic regression equation. The sensitivity of the Spearman rank correlation between the actual and estimated values of the post-release failures yielded a value of 0.686 ($p<0.0005$). The strength of the correlations between the actual and estimated values of the failures indicates the sensitivity of the built models to estimate the post-release failures. The Spearman correlation results between the estimated failure-proneness and actual failures is 0.626, $p<0.0005$. The statistically significant results indicate that with an increase in the estimated failure-proneness there is an increase in the actual failures.

## 5.2 Component B

Component B is part of Windows middleware. Components in the middle layer can be classified broadly as acting as an interface between the users/outer layer and the core/internals of the OS. The component B, in Windows XP-SP1, is of size 1066 KLOC (made up of 1644 files). The generated principal components that account for greater than 95% of the sample variance are shown in Table 8. We use

these two principal components to build our statistical prediction equations.

**Table 8: Principal components for component B**

| Principal Component | Initial Eigen values | | |
|---|---|---|---|
| | Eigen value | % of Variance | Cumulative % |
| PC1 | 4.078 | 67.962 | 67.962 |
| PC2 | 1.794 | 29.905 | 97.867 |

The order of the measures that dominate in the principal component one are the frequency of churn, number of developers the delta lines of code followed by the code complexity. For principal component two, the dominant measures are the arc and block coverage. The $R^2$, Adj. $R^2$, F-test and Nagelkerke $R^2$ indicate the strength and statistical significance of the built multiple and logistic regression models.

$R^2 = 0.573$, Adj-$R^2$=0.502, F=8.045, $p<0.006$; Nagelkerke $R^2 = 0.303$

The sensitivity of the Spearman rank correlation between the actual and estimated values for post-release failures was 0.787 ($p<0.0005$) and between the estimated failure-proneness and actual failures was 0.792 ($p<0.0005$). The results indicate that with an increase in the estimated failures there is an increase in the actual failures; and with an increase in the estimated failure-proneness there is an increase in the actual failures.

### 5.3 Component C

Finally, component C is part of the external layer of the Windows operating systems that deals with UI/customer scenarios and its ability to interact with the middleware interfaces and the inner core components. The extracted code base for Component C for Windows XP-SP1 was 1820 KLOC (consisting of 3265 files). Based on PCA three principal components out of the six generated that account for greater than 95% of the variance as shown in Table 9 are used for building the statistical multiple and logistic regression models.

**Table 9: Principal components for component C**

| Principal Component | Initial Eigen values | | |
|---|---|---|---|
| | Eigen value | % of Variance | Cumulative % |
| PC1 | 4.520 | 75.333 | 75.333 |
| PC2 | .872 | 14.533 | 89.866 |
| PC3 | .333 | 5.555 | 95.421 |

In the extracted principal components, for component 1 all the measures have a substantially equal effect, the order being delta lines of code, frequency of churn, arc coverage, number of users, block coverage and the code complexity. The $R^2$, Adj. $R^2$, F-test and Nagelkerke $R^2$ for the built multiple and logistic regression models are shown below.

$R^2 = 0.665$, Adj. $R^2$=0.626, F=17.188, $p<0.0005$; Nagelkerke $R^2 = 0.635$

The Spearman rank correlation yielded 0.496 ($p<0.0005$). The correlations are statistically significant indicating that with an increase in the actual failures the estimated failures also increase. Similarly the Spearman correlations between the estimated failure-proneness and actual failures is 0.553 ($p< 0.00005$) indicating that with an increase in the estimated failure-proneness there is an increase in the actual failures and vice-versa. This indicates the sensitivity of the build models and in turn the utility of using in-process and product metrics to estimate software quality. Thus the results obtained for components A, B and C *indicate the efficacy of building component-level context-specific models to estimate post-release failures and failure-proneness ($H_3$).*

## 6. Threats to Validity

As stated by Basili et al., drawing general conclusions from empirical studies in software engineering is difficult because any process depends to a large degree on a potentially large number of relevant context variables. For this reason, we cannot assume a priori that the results of a study generalize beyond the specific environment in which it was conducted [3]. Researchers become more confident in a theory when similar findings emerge in different contexts [3]. Since this study was performed on the Windows operating system and the size of the code base and development organization is at a much larger scale than many commercial products, it is likely that the specific models built for Windows would not apply to other products, even those built by Microsoft.

Considering the highly sensitive nature of this data we do not present graphical plots of the actual and estimated failures (or) failure-proneness. Techniques like statistical correlation between the actual and estimated values indicating the sensitivity of prediction show the strength of the results with a reduced degree of accuracy. The important underlying contribution here we would like to indicate is the utility of using in-process and product metrics from large software systems to make early estimates of software quality in terms of failures/failure-proneness.

## 7. Conclusions and Future Work

Fenton et al. [14] note the need for empirical studies in terms of case studies, surveys and experiments to: *confirm a theory; and to explore a relationship*. We investigated the *theory* that in-process and product metrics can be utilized early indicators of software failures/failure-proneness before the software system actually ships. For this purpose we utilize the *relationship* amongst these product and process measures to explain and build our statistical prediction equations.

In general it is beneficial to obtain early estimates of failure-proneness/failures to help inform decisions on testing, code inspections, design rework, as well as financial costs associated with a delayed release. From our statistical analysis in Section 4 and Section 5 we observe that this is possible using historical in-process and product metrics. The fit of our built multiple and logistic regression models in conjunction with the sensitivity of our predictions indicate that,

- Historical in-process and product metrics from Windows XP-SP1 can be used to estimate the post-release failures at statistically significant levels for Windows Server 2003.
- Historical in-process and product metrics from Windows XP-SP1 can be used to identify failure-prone modules for Windows Server 2003 at statistically significant levels.
- Component-level context-specific statistical models can be used to estimate post-release failures and failure-proneness at statistically significant levels.

We also intend that our case study contributes towards strengthening the existing empirical body of knowledge in this field [10, 13, 15, 18, 20, 27, 28] for early estimation of software quality using in-process and product metrics.

In the future, as part of the MetriZone project we will integrate various other product complexity metrics, architectural dependences etc. to identify the best set of predictors for estimating software quality. We are currently building such models for Windows Vista, Microsoft's next generation operating system. We intend to build context specific robust prediction equations for all components in Windows. We also plan to replicate our study with other Microsoft products like SQL Server, Microsoft OFFICE etc.

## References


[1] "IEEE Std 982.2-1988 IEEE guide for the use of IEEE standard dictionary of measures to produce reliable software," 1988.

[2] V. Basili, Briand, L., Melo, W., "A Validation of Object Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, 22(10), pp. 751 - 761, 1996.

[3] V. Basili, Shull, F.,Lanubile, F., "Building Knowledge through Families of Experiments", *IEEE Transactions on Software Engineering*, 25(4), pp.456-473, 1999.

[4] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.

[5] L. C. Briand, Wuest, J., Daly, J.W., Porter, D.V., "Exploring the Relationship between Design Measures and Software Quality in Object Oriented Systems", *Journal of Systems and Software*, 51(3), pp. 245-273, 2000.

[6] L. C. Briand, Wuest, J., Ikonomovski, S., Lounis, H., "Investigating quality factors in object-oriented designs: an industrial case study", Proceedings of International Conference on Software Engineering, pp. 345-354, 1999.

[7] F. P. Brooks, *The Mythical Man-Month, Anniversary Edition*: Addison-Wesley Publishing Company, 1995.

[8] S. R. Chidamber, Kemerer, C.F., "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, 20(6), pp. 476-493, 1994.

[9] G. Denaro, Morasca, S., Pezze., M, "Deriving models of software fault-proneness", Proceedings of International Conference on Software Engineering Knowledge Engineering, pp. 361-368, 2002.

[10] G. Denaro, Pezze., M, "An empirical evaluation of fault-proneness models", Proceedings of International Conference on Software Engineering, pp. 241-251, 2002.

[11] K. El Emam, "A Methodology for Validating Software Product Metrics," National Research Council of Canada, Ottawa, Ontario, Canada NCR/ERC-1076, June 2000.

[12] N. Fenton, Neil, M., "A critique of software defect prediction models", *IEEE Transactions on Software Engineering*, 25(5), pp. 675-689, 1999.

[13] N. E. Fenton, Ohlsson, N., "Quantitative analysis of faults and failures in a complex software system", *IEEE Transactions on Software Engineering*, 26(8), pp. 797-814, 2000.

[14] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*: Brooks/Cole, 1998.

[15] T. L. Graves, Karr, A.F., Marron, J.S., Siy, H., "Predicting Fault Incidence Using Software Change History", *IEEE Transactions on Software Engineering*, 26(7), pp. 653-661, 2000.

[16] ISO/IEC, "DIS 14598-1 Information Technology - Software Product Evaluation", 1996.

[17] E. J. Jackson, *A Users Guide to Principal Components*. Hoboken, NJ: John Wiley & Sons Inc., 2003.

[18] T. M. Khoshgoftaar, Allen, E.B., Goel, N., Nandi, A., McMullan, J., "Detection of Software Modules with high Debug Code Churn in a very large Legacy System", Proceedings of International Symposium on Software Reliability Engineering, pp. 364-371, 1996.

[19] T. M. Khoshgoftaar, Allen, E.B., Jones,W.D., Hudepohl, J.P., "Classification-Tree Models of Software Quality Over Multiple Releases", *IEEE Transactions on Reliability*, 49(1), pp. 4-11, 2000.

[20] T. M. Khoshgoftaar, Szabo, R.M., "Improving Code Churn Predictions During the System Test and Maintenance Phases", Proceedings of IEEE International Conference on Software Maintenance, pp. 58-67, 1994.

[21] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, 2(4), pp. 308-320, 1976.

[22] A. Mockus, Zhang, P., Li, P., "Drivers for customer perceived software quality", Proceedings of International Conference on Software Engineering, pp. 225-233, 2005.

[23] N. Nagappan, Ball, T., "Static Analysis Tools as Early indicators of Pre-Release Defect Density", Proceedings of International Conference on Software Engineering, pp. 580-586, 2005.

[24] N. Nagappan, Ball, T., "Use of Relative Code Churn Measures to Predict System Defect Density", Proceedings of International Conference on Software Engineering, pp. 284-292, 2005.

[25] N. Nagappan, Ball, T., Zeller, A., "Mining metrics to predict component failures", Proceedings of International Conference on Software Engineering, pp. 452-461, 2006.

[26] N. J. D. Nagelkerke, "A note on a general definition of the coefficient of determination", *Biometrika*, 78(3), pp. 691-692, 1991.

[27] M. C. Ohlsson, von Mayrhauser, A., McGuire, B., Wohlin, C., "Code Decay Analysis of Legacy Software through Successive Releases", Proceedings of IEEE Aerospace Conference, pp. 69-81, 1999.

[28] T. J. Ostrand, Weyuker, E.J, Bell, R.M., "Where the Bugs Are", Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 86-96, 2004.

[29] R. Subramanyam, Krishnan, M.S., "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", *IEEE Transactions on Software Engineering*, Vol. 29(4), pp. 297 - 310, 2003.

[30] M.-H. Tang, Kao, M-H., Chen, M-H., "An empirical study on object-oriented metrics", Proceedings of Sixth International Software Metrics Symposium, pp. 242-249, 1999.

[31] T. Zimmermann, Weißgerber, P., Diehl, S., Zeller, A., "Mining Version Histories to Guide Software Changes", *IEEE Transactions in Software Engineering*, 31(6), pp. 429-445, 2005.