Using In-Process Testing Metrics to Estimate Post-Release Field Quality of Java Programs

Nachiappan Nagappan[1], Laurie Williams[2], Mladen Vouk[2], Jason Osborne[2]
[1] *Microsoft Research, Redmond, WA 98052*
nachin@microsoft.com
[2] *North Carolina State University, Raleigh, NC 27695*
lawilli3, vouk, jaosborn,@ncsu.edu

Abstract

In industrial practice, information on the software field quality of a product is available too late in the software lifecycle to guide affordable corrective action. An important step towards remediation of this problem lies in the ability to provide an early estimation of post-release field quality. This paper presents a suite of nine static in-process unit test metrics, the Software Testing and Reliability Early Warning for Java (STREW-J) metric suite. The STREW-J metrics can be used to leverage the software testing effort to predict post-release field quality early and throughout the software development phases. The metric suite is applicable for software products implemented in Java for which an extensive suite of automated unit test cases are incrementally created as development proceeds. We built and validated a prediction model using the STREW-J metrics via a three-phase case study approach which progressively involved 22 small-scale academic projects, 27 medium-sized open source projects, and five industrial projects. The ability of the STREW-J metric suite to estimate post-release field quality via a statistical regression model was evaluated in the three different environments. The error in estimation and the sensitivity of the predictions indicate the STREW-J metric suite can be used effectively to predict post-release software field quality.

1. Introduction

In industry, estimates of software field quality are often available too late in the software lifecycle to guide affordable corrective action to the quality of the software. Field quality cannot be measured before a product has been completed and delivered to an internal or external customer. Because this information is available late in the software lifecycle, corrective actions tend to be expensive [3]. Software developers can benefit from an early warning regarding the quality of their product.

In our research, we formulate this early warning from a collection of static automated unit test metrics that are correlated with Trouble Reports (TRs) per thousand lines of code (KLOC), an external measure of quality obtained from users. A TR [25] is a customer-reported problem whereby the software system does not behave as the customer expects. An internal metric, such as cylomatic complexity [24], is a measure derived from the product itself [16]. An external measure is a measure of a product derived from the external assessment of the behavior of the system [16]. 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 [15] 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 [12].

Our research objective is to construct and validate a set of easy-to-measure, in-process, internal, static unit test metrics that can be used as an early indication of the external measure post-release field quality. To this end, we have created a metric suite we call the Software Testing and Reliability Early Warning for Java (STREW-J) metric suite [26]. As will be discussed, the metric suite is applicable for software products implemented in Java for which an extensive suite of automated unit test cases are incrementally created as development proceeds. Metrics about these automated unit tests are used to predict post-release field quality and to provide an indication of whether the composition of the unit test cases is adequate to yield a high quality product based upon past results. The STREW-J metrics are used to build a regression

model to estimate the post-release field quality in terms of the metric TRs/KLOC.

Teams develop a history of the value of the STREW-J metrics from comparable projects with acceptable levels of field quality. These historical metric values are then used to estimate the relationship between the STREW-J metric elements and the TRs/KLOC. This early estimate helps to identify areas that require more testing, based on the estimates of post-release field quality.

In this paper, we present empirical results of an academic feasibility study (22 projects), a case study of open source projects (27 projects), and an industrial case study (five projects) which were used to build and validate the STREW-J model as a means of estimating post-release field quality. The remainder of this paper is organized as follows. Section 2 discusses the related work. Section 3 presents the STREW-J metric suite. Section 4 discusses the model building and validation and Section 5 the case studies. Section 6 presents the conclusions.

2. Related Work

Research shows that internal metrics can be related to external measures of field quality, such as fault-proneness. Software fault-proneness is defined as the probability of the presence of faults in the software [10]. 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 [10]. Structural object-orientation (O-O) measurements, such as those in the Chidamber-Kemerer (C-K) O-O metric suite [7], have been used to evaluate and predict fault-proneness [1, 4, 5]. The CK metric suite consists of six metrics: weighted methods per class (WMC), coupling between objects (CBO), depth of inheritance tree (DIT), number of children (NOC), response for a class (RFC) and lack of cohesion among methods (LCOM). These metrics have been shown to be useful early internal indicators of externally-visible product quality in terms of fault-proneness [1, 32].

Basili et al. [1] studied the fault-proneness in software programs using eight student projects. They observed that the WMC, CBO, DIT, NOC and RFC were correlated with defects while the LCOM was not correlated with defects. Further, Briand et al. [5] performed an industrial case study and observed the CBO, RFC, and LCOM to be associated with the fault-proneness of a class. A similar study done by Briand et al. [4] on eight student projects showed that classes with a higher WMC, CBO, DIT and RFC were more fault-prone while classes with a higher NOC were less fault-prone. Tang et al. [33] studied three real time systems for testing and maintenance defects. Higher WMC and RFC were found to be associated with fault-proneness. El Emam et al. [13] studied the effect of class size on fault-proneness by using a large telecommunications application. Class size was found to confound the effect of all the metrics on fault-proneness. Finally, Chidamber et al. [6] analyzed project productivity, rework, and design effort of three financial services applications. High CBO and low LCOM were associated with lower productivity, greater rework, and greater design effort.

Vouk and Tai [35] showed that in-process metrics have strong correlation with field quality of industrial software products. They demonstrated the use of software metric estimators, such as the number of failures, failure intensity (indicated by failures per test case), and drivers such as change level measured by lines of code, component usage, and effort to:

1. quantify component quality in terms of the number of failures expected during initial operational deployment;
2. identify fault-prone and failure-prone components; and
3. guide the software testing process to minimize the number of failures/faults that can be expected in future phases.

To summarize, there is a growing body of empirical evidence that supports the theoretical validity of the use of these internal metrics as predictors of fault-proneness. The consistency of these findings varies with the programming language [32]. Therefore, the metrics are still open to debate [8]. The STREW-J will be used to predict the external measure of field quality.

3. STREW Metric Suite

The STREW-J metric suite consists of internal, in-process unit test metrics that are leveraged to estimate post-release field quality with an associated confidence interval. Section 3.1 discusses the applicability of the STREW-J metric suite, and Section 3.2 the STREW-J metric suite elements.

3.1 Applicability

The use of the STREW-J metrics is predicated on the existence of an extensive suite of automated unit test cases being created as development proceeds. These automated unit tests need to be structured as is done with the one of the object-oriented (O-O) xUnit[1] testing frameworks, such as JUnit[2]. The STREW-J

[1] http://xprogramming.com/software.htm
[2] http://junit.org/

method is not applicable for script-based automated testing because, as will be discussed, the metrics are primarily based upon the O-O programming paradigm. When these xUnit frameworks are used with O-O programming, both test code and implementation code hierarchies emerge.

Figure 1 presents a simplistic example of a parallel structure between a source and test class. For each implementation source code class (e.g. `computation`), there exists a corresponding test code class (e.g. `computationTest`). Often each method/function in an implementation source code class (e.g. `addi`) will have one or more corresponding test code method(s)/functions(s) (e.g. `testaddi`). In industrial practice, often such perfect parallel class structure and one-to-one method/function correspondence is not observed; our example is overly simplistic for illustrative purposes. However, a test hierarchy which ultimately inherits from the TestCase class (in JUnit) is created to exercise the implementation code.

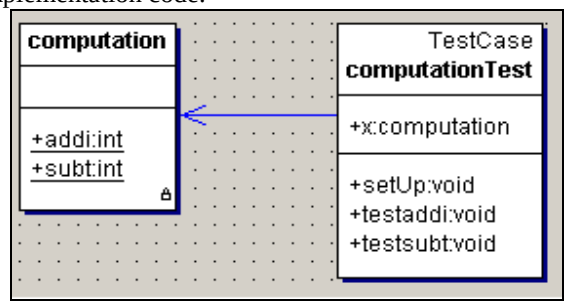

Figure 1: Corresponding source and test classes

Table 1 provides sample Java code for a computation class which adds and subtracts two integers. Notice the `assertEquals` keyword in the `testaddi` and `testsubt` methods. xUnit testing is assert-based. The number of asserts [31] and other metrics specific to xUnit-like test automation are used in the STREW-J metric suite.

Table 1: Corresponding source and test code

```
public class computation
{
   public static int addi(int t1,int t2)
{
              int t3
              temp3 = t1 + t2;
              return(t3);
    }
   public static int subt(int t1,int t2)
{
              int t3
              t3 = t1 - t2;
              return(t3);
    }
}
```

```
public class computationTest extends
TestCase
{
       public computation x;
       public void setUp() {
                x=new computation();
        }
    public void testaddi() {

    assertEquals(100,x.addi(75,25));
    }
    public void testsubt() {

    assertEquals(40,x.subt(50,10));
    }
}
```

3.2 STREW-J Metric Suite Elements

The STREW-J Version 2.0 metric suite consists of nine constituent metric ratios. The metrics are intended to cross-check each other and to triangulate upon an estimate of post-release field quality. Each metric makes an individual contribution toward estimation of the post-release field quality but work best when used together. Development teams record the values of these nine metrics and the actual TRs/KLOC of projects. The historical values from prior projects are used to build a regression model that is used to estimate the TRs/KLOC of the current project under development.

The nine constituent STREW-J metrics (SM1 – SM9) and instructions for data collection and computation are shown in Table 2. The nine metrics are static unit test metrics and do not include dynamic metrics, such as code coverage. The metrics can be categorized into three groups: test quantification metrics, complexity and O-O metrics, and a size adjustment metric.

The **test quantification metrics** (SM1, SM2, SM3, and SM4) are specifically intended to crosscheck each other to account for coding/testing styles. For example, one developer might write fewer test cases, each with multiple asserts checking various conditions. Another developer might test the same conditions by writing many more test cases, each with only one assert. We intend for our metric suite to provide useful guidance to each of these developers without prescribing the style of writing the test cases. Assertions are used in two of the metrics as a means for demonstrating that the program is behaving as expected and as an indication of how thoroughly the source classes have been tested on a per class level. SM4 serves as a control measure to counter the confounding effect of class size, as shown by El-Emam [13], on the prediction efficiency.

Table 2: STREW-J metrics and collection and computation instructions

Test quantification	
Metric	**ID**
$\frac{\text{Number of Assertions}}{SLOC^{*}}$	SM1
$\frac{\text{Number of Test Cases}}{SLOC^{*}}$	SM2
$\frac{\text{Number of Assertions}}{\text{Number of Test Cases}}$	SM3
$\frac{(TLOC^{+}/SLOC^{*})}{(\text{Number of Classes}_{Test} / \text{Number of Classes}_{Source})}$	SM4
Complexity and O-O metrics	
$\frac{\Sigma\ \text{Cyclomatic Complexity}_{Test}}{\Sigma\ \text{Cyclomatic Complexity}_{Source}}$	SM5
$\frac{\Sigma\ CBO_{Test}}{\Sigma\ CBO_{Source}}$	SM6
$\frac{\Sigma\ DIT_{Test}}{\Sigma\ DIT_{Source}}$	SM7
$\frac{\Sigma\ \text{WMC}_{Test}}{\Sigma\ \text{WMC}_{Source}}$	SM8
Size adjustment	
$\frac{SLOC^{*}}{\text{Minimum } SLOC^{*}}$	SM9
* *Source Lines of Code (SLOC) is computed as non-blank, non-comment source lines of code* + *Test Lines of Code (TLOC) is computed as non-blank, non-comment test lines of code*	

The **complexity and O-O metrics** (SM5, SM6, SM7, and SM8) examine the relative ratio of test to source code for control flow complexity and for a subset of the CK metrics. The dual hierarchy of the test and source code allows us to collect and relate these metrics for both test and source code. These relative ratios for a product under development can be compared with the historical values from prior comparable projects to indicate the relative complexity of the testing effort with respect to the source code. The metrics are now discussed more fully:

- The *cyclomatic complexity* [24] metric for software systems is adapted from the classical graph theoretical cyclomatic number and can be defined as the number of linearly independent paths in a program. Prior studies have found a strong correlation between the cyclomatic complexity measure and the number of test defects [34]. Studies have also shown that code complexity correlates strongly with program size measured by lines of code [19] and is an indication of the extent to which control flow is used. The use of conditional statements increases the amount of testing required because there are more logic and data flow paths to be verified [20].
- The larger the inter-object *coupling (CBO)*, the higher the sensitivity to change [7]. Therefore, maintenance of the code is more difficult [7]. Prior studies have shown coupling to be related to fault-proneness [1, 4, 5]. As a result, the higher the inter-object class coupling, the more rigorous the testing should be [7].
- A higher depth of inheritance (*DIT)* indicates desirable reuse but adds to the complexity of the code because a change or a failure in a super class propagates down the inheritance tree. DIT and fault-proneness were shown to be strongly correlated [1, 4].
- The *number of methods* and the *complexity of methods* is a predictor of how much time and effort is required to develop and maintain the class [7]. The larger the number of methods in a class, the greater is the potential impact on children, since the children will inherit all the methods defined in the class. The ratio of the WMC_{test} and $\text{WMC}_{\text{source}}$ measures the relative ratio of the number of test methods to source methods. This measure serves to compare the testing effort on a method basis. The relationship between the WMC as an indicator of fault-proneness has been demonstrated in prior studies [1, 5].

The final metric is a **relative size adjustment factor (RSAF)**. Defect density has been shown to increase with class size [13]. We account for project size in terms of SLOC for the projects used to build the STREW-J prediction equation using RSAF.

The metrics that comprise the STREW-J metric suite have evolved [27, 30] through our case studies. Some metrics were removed based on the lack of their ability to contribute to the estimation of post-release field quality and due to pre-existing inter-correlations between the elements. These metrics were removed based upon statistical inter-correlations, multicollinearity, stepwise, backward, and forward regression techniques [23] of case study data. The following metrics were removed:

- Number of requirements/Source lines of code
- Number of $\text{children}_{\text{test}}$/Number of $\text{children}_{\text{source}}$
- Lack of cohesion among $\text{methods}_{\text{test}}$/Lack of cohesion among $\text{methods}_{\text{source}}$

4. STREW-J MODEL BUILDING AND VALIDATION

This section discusses the three-phase of case studies used to build and validate the use of the STREW-J metric suite for estimating post-release field quality. The validation studies are performed in three

different environments to build an empirical body of knowledge. 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 [2]. Researchers become more confident in a theory when similar findings emerge in different contexts [2]. By performing multiple case studies and/or experiments and recording the context variables of each case study, researchers can build up knowledge through a family of experiments [2] which examine the efficacy of a new practice. Replication of experiments addresses threats to experimental validity. We address these issues related to empirical studies by replicating multiple case studies through a family of experiments in three different contexts: academic, open source and industrial. Similar results in these contexts indicate the promise of this approach at statistically significant levels.

4.1 Model Building

This section describes the model building strategies that were used for predicting post-release field quality. A number of techniques have been used for the analysis of software quality. Multiple linear regression (MLR) analysis [21] has been used to model the relationship between quality and software metrics. The general regression equation is of the form:

$$Y = c + a_1X1 + a_2X2 + \ldots\ldots + a_nXn, \quad (1)$$

where Y is the dependent variable, a_1, a_2 …., a_n are regression coefficients and X1, X2 …., Xn are the known independent variables.

In our work, the dependent variable to be estimated is the post-release field quality and the independent variables are the STREW-J metrics. A difficulty associated with MLR is multicollinearity among the metrics that can lead to inflated variance in the estimation of reliability. One approach that has been used to overcome this difficulty is Principal Component Analysis (PCA) [17]. With PCA, a smaller 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 [11] can be built to model data using the principal components as the independent or predictor variables. Denaro et al. [11] calculated 38 different software metrics for the open source Apache 1.3 and 2.0 projects. Using PCA, they selected a subset of nine of these metrics that explained 95% of the total sample variance.

As will be discussed, in our case studies the nine STREW-J metrics demonstrated multicollinearity, and therefore PCA was used to reduce the STREW-J metrics that are highly correlated with other metrics to a smaller number of uncorrelated linear combinations of these metrics. The Kaiser-Meyer-Olkin (KMO) [18] test of sampling adequacy was used to identify multicollinearity, indicating the applicability of using the principal components produced by PCA, rather than the nine individual metrics, in an MLR equation. PCA generates uncorrelated linear combinations of attributes, overcoming the variance inflation in estimating post-release field quality caused by multicollinearity. Multiple regression analysis is performed on the principal components produced. The generalized regression equation and associated confidence are formulated as follows:

$$Y_{new} = c + a_1PC1 + a_2PC2 + \ldots\ldots + a_nPCn, \quad (2)$$

where a_1, a_2 …., a_n are regression coefficients and PC1, PC2 …., PCn are the produced principal components.

The estimated value of post-release field quality (Y_{new}) is calculated with an associated confidence bound [23] given by Equation 3 where *n* is the number of observations, $t_{(\alpha/2,\ n)}$ is the standard t-table value with n degrees of freedom at a significance level of $\alpha = 0.05$.

$$\textit{Confidence bounds} = Y_{new} \pm t_{(\alpha/2,\ n)} * \textit{Mean Standard Error} \quad (3)$$

Further, the model building strategies have the following associated factors:

- The coefficient of determination, R^2, 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 is to test the null hypothesis that all regression coefficients are zero at statistically significant levels.
- Where parametric testing is appropriate, a significance level of $\alpha = 0.05$ was adopted for statistical inference. For example, all interval estimates are reported using confidence level 95% ($\alpha = 0.05$).

4.2 Results Evaluation Techniques

The evaluation of the TRs/KLOC estimates with the actual values is performed using two methods explained below:

Average Absolute Error (AAE) and Average Relative Error (ARE). AAE and ARE are used to measure the accuracy of prediction of the estimated TRs/KLOC with numerical quantification (i.e. numerical accuracy) [22]. The smaller the values of AAE and ARE, the better are the predictions. The equations for AAE and ARE are provided as Equations 4 and 5 where *n* is the number of observations.

$$AAE = \frac{1}{n}\sum_{i=1}^{n} |Estimated\ Value - Actual\ Value| \quad (4)$$

$$ARE = \frac{1}{n}\sum_{i=1}^{n} \frac{|\ Estimated Value - Actual Value}{Actual Value} \quad (5)$$

Correlation between the actual and estimated TRs/KLOC. This correlation is used to quantify the sensitivity of prediction. The two correlation techniques utilized are (1) the Spearman rank correlation which is a commonly-used robust rank correlation technique because it can be applied when the association between elements is non-linear; and (2) the Pearson bivariate correlation is best suited for normally distributed data and where the association between elements is linear [14]. A positive relationship between the actual and estimated values is desired in terms of the more robust Spearman rank correlation.

A positive correlation between the actual and estimated TRs/KLOC would indicate that with an increase in estimated TRs/KLOC, there is an increase in actual TRs/KLOC. A negative correlation would indicate that with an increase in estimated TRs/KLOC, there is a decrease in actual TRs/KLOC (or vice versa).

5 Case Studies

This section describes the three-phase empirical case study approach used to build the STREW-J model and to validate the effectiveness of the model. The three phases include case studies of 22 academic projects, 27 open source projects, and five industrial projects. For our case studies, the TRs were screened to remove duplicates and reports due to documentation problems. The data used for the academic and open source projects are publicly available in [26].

5.1 Feasibility Study – Academic projects

A controlled feasibility study was performed with junior/senior-level students at North Carolina State University (NCSU) to investigate the efficacy of the STREW-J metric suite elements to estimate post-release field quality. The students worked on a project that involved development of an Eclipse[3] plug-in to collect software metrics. The project was six weeks in duration and used Java as the programming language. The JUnit testing framework was used for unit testing; students were required to have 80% statement coverage. A total of 22 projects were submitted, and each group had four to five students. The projects were between 617 LOC_{source} and 3,631 LOC_{source}. On average, the ratio of LOC_{test} to LOC_{source} was 0.35. Each project was evaluated by 45 independent test cases. Post-release field quality was estimated by instructor-run black box test case failures per KLOC because the student projects were not released to customers. If a 2 KLOC project had ten failures (out of the 45 tests) the TRs/KLOC = (10/2) = 5 failures/KLOC. The regression model was built using the principal components obtained from the STREW-J metrics as the independent variables and the post-release field quality as the dependent variable for all 22 projects. The built model had an R^2 value = 0.512 and was statistically significant (F=6.284, p=0.004).

To evaluate the efficacy of the STREW-J metric suite to predict TRs/KLOC, we use a random data splitting approach. We randomly selected two-thirds (N=15) of the samples to build a prediction model and the remaining one-third (N=7) to evaluate the built model. The resulting model had an R^2= 0.598, (F=5.463, p=0.015). The AAE value was 4.47 and ARE was 1.27 respectively. A repeated random splitting yielded results [29]. The results of this feasibility study motivated further investigation using open source and industrial projects to investigate the utility of the STREW-J metric suite to predict TRs/KLOC on larger scale projects. The detailed results of the feasibility study published earlier is available here [29].

The academic study is a closed structured environment that might represent ideal conditions but may not reflect industrial settings. As a result, an external validity issue arises because the programs were written by students, and the projects are small relative to industry y applications. This concern is mitigated to some degree because the students were advanced undergraduates (junior/senior). Another threat to validity is that using the same set of 45 test cases to evaluate post-release field quality may also have skewed the failures/KLOC to a certain degree. It is possible that under exhaustive usage by customers a project with higher 'test' failures/KLOC would actually have fewer customer-realized failures.

5.2 Open source case studies

Open source projects are convenient to perform post-release field quality analysis because source code, test code, and defect logs are openly available for use. Additionally, these projects are more representative of industrial projects than academic projects due to their size and scope. We selected 27 open source projects to apply the STREW-J metric suite.

Description: Twenty-seven open source projects that were developed in Java were selected from Sourceforge (http://sourceforge.net). The following criterion was used to select the projects from Sourceforge.

- *software development tools*. All of the chosen projects are software development tools, i.e. tools

[3] http://eclipse.org/

that are used to build and test software and to detect defects in software systems to select programs from a common domain.

- *download ranking of 85% or higher.* In Sourceforge, the projects are all ranked based on the number of times they have been downloaded on a percentile scale from 0-100%. For example, a ranking of 85% means that a product is in the top 15% in terms of download quantity. We chose this criterion because we reasoned that a comparative group of projects with similarly high download rates would be more likely to have a similar usage frequency by customers that would ultimately reflect the post-release field quality.
- *automated unit testing*. The projects needed to have JUnit automated tests.
- *defect logs available*. The defect log needed to be available for identifying TRs with the date of the TR reported.
- *active fixing of TRs*. The TR fixing rate is used to indicate the system is still in use. The time between the reporting of a TR and the developer fixing it serves as a measure of this factor. Projects that had open TRs that were not assigned to anyone over a period of three months were not considered.
- *Sourceforge development stage of 4 or higher*. This denotes the development stage of the project (1-6) where 1 is a planning stage and 6 is a mature phase. We chose a cut-off of 4 which indicates the project is at least a successful beta release. This criterion indicates that the projects are at a similar stage of development and are not projects too early in the development lifecycle.

Figure 2 shows the names of the projects and their sizes in LOC_{source}. On average, the ratio of LOC_{test} to LOC_{source} was 0.37. The projects range from around 2.5 $KLOC_{source}$ to 80 $KLOC_{source}$. The TRs are normally distributed with a range from 0.20 to 6.9 TRs/KLOC (Mean = 1.42). The defect logs were screened, and duplicate TRs were removed to obtain an accurate measure of the TRs/KLOC.

Limitations: There is a validity issue with respect to the actual usage of open source software systems. The commonly-available usage metric for open source software is the download activity ratio. For example, it might be possible that both Project 1 and Project 2 could have both been downloaded by 100 users, but Project 1 might have been actually used by only ten users while Project 2 by 90 users. Actual usage is not directly observable from the sourceforge.com. Also, we are dependent on customer reports for measuring the TRs/KLOC. However, some customers might not report the TRs in the open source environment.

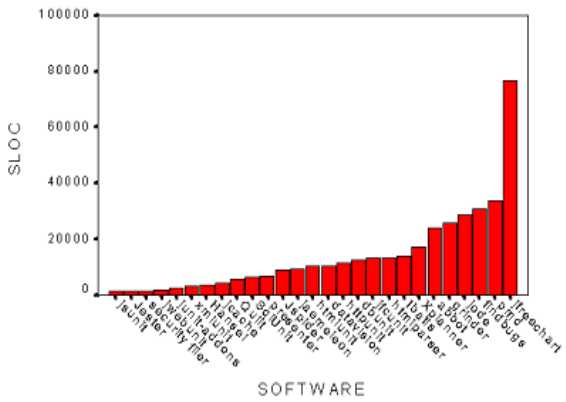

Figure 2: Open source project sizes

We assume that the TRs are representative of the operational profile [29] of the software system. If the different components of the software systems were not used equally, the estimation of TRs can be misleading. For example, a different set of customers could use the product for a different purpose and might exercise the product in a different way. This different usage would exercise different parts of the code and could surface a different post-release product quality. The issues of the generalizability of the actual operation profile is negated to some extent by the uniform testing efforts of the projects (using unit testing measured by test lines of code, assertions, and test cases), comparable TRs/KLOC across all the projects, and all the projects belonging to one particular domain of software systems.

Model building: Multiple regression was performed using the principal components of the STREW-J metrics for the 27 open source projects as the independent variable and the post-release field quality as the dependent variable. Similar to the academic case studies, the model was statistically significant (R^2=0.428, F=8.993, p=0.004) in its ability to explain the variance in the dependent variable.

The Kaiser-Meyer-Olkin (KMO) measure of sampling adequacy is 0.764 indicating the applicability of using PCA in model building. The PCA of the nine STREW-J metrics yields two principal components (with eigen values greater than 1). The linear transformation coefficient for the two principle components is given by the component matrix in Table 3.

Using MLR on the two principal components calculated using Table 3, we obtain Equation 6.

*Post-release field quality = 1.424 + 0.852 * PC1 - 0.132*PC2* (6)

where PC1 and PC2 are the transformed principal components produced from the nine STREW-J metrics.

Table 3: Component matrix for PCA transformation

	Component	
	PC1	PC2
SM1	.793	.284
SM2	.795	-.281
SM3	-.270	.701
SM4	.935	-6.280E-03
SM5	.159	-.745
SM6	.877	.102
SM7	.699	.457
SM8	.864	4.126E-02
SM9	-.569	.233

Random data splitting: For the random data-splitting, we use two thirds of the projects (N=18) to build the prediction model and the remaining one-third (N=9) to evaluate the fit of the prediction model. We repeated the random split nine times to verify data consistency, i.e. to check if the results of our analysis were not a one-time occurrence.

To summarize, we present the results of our evaluation using the models built using PCA in Table 4. We can assess the efficacy of the prediction model built using 18 randomly-chosen projects. The ARE values that reflect the relative error in terms of the absolute magnitude of the TRs/KLOC. The overall standard deviation of the TRs/KLOC is 1.318 TRs/KLOC.

The AAE in all the nine random cases (using PCA after eliminating multicollinearity) is smaller than the standard deviation, indicating that the efficacy of the prediction results.

Table 4 indicates the correlation coefficient (Pearson and Spearman) results between the actual and estimated post-release TRs/KLOC. The correlation measure serves to indicate the sensitivity between the actual and predicted post-release TRs/KLOC. Of nine random samples in the PCA, seven are statistically significant (between the estimated and actual post-release quality), indicating the efficacy of our approach for the open source projects case study (shown in bold in Table 4).

5.3 Industrial case study

In this section we describe the industrial case study that was performed with a multinational, travel services company to investigate the efficacy of the STREW-J metric suite to assess post-release field quality for three commercial software systems.

Description: Our industrial case study involved three software systems (five versions). To protect proprietary information, we keep the name and nature of the projects anonymous. These projects were critical in nature because failures could lead to loss of essential funds for the company. The project sizes that were used for analysis are shown in Table 5. The development language used was Java, and the JUnit testing framework was used for unit and acceptance testing.

Table 5: Industrial project sizes

Project	Size
Project 1A	190 KLOC
Project 1B	193 KLOC
Project 2A	504 KLOC
Project 2B	487 KLOC
Project 3	13 KLOC

Limitations: Project 3 was of much smaller size compared to the other projects. Also Project 3 was a new release of a system. But all the three projects belonged to the same domain in terms of functional usage thereby having comparable TR/KLOC values.

Post-release field quality prediction: In the industrial environment, failures found by customers are reported back to the organization. These failures are then mapped back to the appropriate software systems. We use Equation 6, built using the principal components of the STREW-J metrics of the open source projects, to predict the post-release field quality of the industrial software systems. Figure 3 indicates the prediction plots obtained using PCA.

Table 4: AAE and ARE for PCA models

Random Split	AAE	ARE	Pearson (p value)	Spearman (p value)
1.	1.1230	0.3276	0.834 (p=0.005)	**0.700** (p=0.036)
2.	0.6400	0.2796	0.694 (p=0.038)	**0.617** (p=0.077)
3.	0.5600	0.2900	0.641 (p=0.063)	**0.667** (p=0.050)
4.	0.7322	0.4034	0.732 (p=0.025)	**0.700** (p=0.036)
5.	0.5533	0.2599	0.799 (p=0.010)	**0.717** (p=0.030)
6.	0.6200	0.2900	0.279 (p=0.467)	0.317 (p=0.406)
7.	0.3555	0.1525	0.905 (p=0.001)	**0.917** (p=0.001)
8.	1.1011	0.574	0.351 (p=0.354)	0.467 (p=0.205)
9.	0.4922	0.2493	0.721 (p=0.028)	**0.667** (p=0.050)

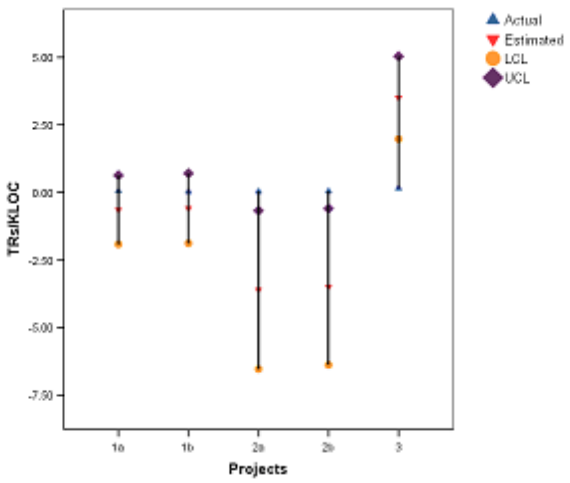

Figure 3: Prediction plots with PCA

Figure 3 indicates the predicted values of the actual TRs/KLOC and bounds for Projects 1A/B, 2A/B and C. The actual normalized TRs/KLOC values are close to the zero for all the projects. The critical nature of the project drives the development organization to focus on practices which yield high quality products. The lower confidence bounds (LCL) are all negative (although it is impossible to actually have a negative TR/KLOC) because the regression equation yields point estimates which appropriately predict a near-perfect project. Project 3 exhibits a variant behavior than the others. Project 3 may not be a comparable project because it is smaller than the other projects and was to form the core components of the organization's future software products. Therefore, Project 3 was particularly well tested, considerably more than the other software systems used to build or evaluate the prediction. On closer observation of the nine STREW-J metric values for Project 3, we notice that metrics SM1, SM2 for Project 3 at least one order greater than all the projects (open source included). Metrics SM5, SM6, SM7, and SM8 for Project 3 are at least twice of SM5, SM6, SM7 and SM8 of all the other projects. SM5-SM8 indicate that Project 3 had a high testing effort (and complexity) compared to all the other projects. These differences prevent the open source model from being as accurate a predictor for Project 3.

In all cases, the upper confidence bounds (UCL) are larger than the actual value. Organizations can take a conservative approach and use the upper bounds of TRs/KLOC to be the actual estimated TRs/KLOC to drive the overall quality higher. Even though the sample size is small (only 5 projects), we present the correlation results of the actual and estimated TRs/KLOC indicated in Figure 3. The Pearson correlation coefficient = 0.962 ($p = 0.009$) and Spearman correlation coefficient = 0.500 ($p = 0.391$). This indicates the efficacy of the sensitivity of prediction between the actual and estimated TRs/KLOC, but is limited to a certain degree by the small sample size of the available projects.

7. Conclusions and Future Work

Feedback on important attributes of a software testing effort can be useful to developers because it helps identify weaknesses and the completeness of testing phase. Our prior work on using STREW metrics addressed this issue [28]. In this paper we have reported on the use of the STREW measures for providing such a test quality feedback in a controlled academic, open source and industrial environment. The results indicate the efficacy of the STREW metric suite to provide meaningful feedback on the quality of the testing effort.

Further, for providing test quality feedback, we have automated the collection and analysis of statement and branch coverage and an earlier version of the STREW metrics suite via an open source Eclipse plug-in GERT (Good Enough Reliability Tool) [9]. We are updating the tool to reflect the current version of the STREW metric suite. We also plan to use the test quality feedback standards in-process in industrial organizations and to study the benefits of early feedback on the quality of the testing effort.

References

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

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

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

[4] L. Briand, J. Wuest, J. Daly, and V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems ", *Journal of Systems and Software*, 51(3), pp. 245-273, 2000.

[5] L. C. Briand, J. Wust, and H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: An Industrial Case Study", Proceedings of International Conference on Software Engineering, pp. 345-354, 1998.

[6] S. R. Chidamber, D. P. Darcy, and C. F. Kemerer, "Managerial Use of Object-Oriented Software: An Explanatory Analysis", *IEEE Transactions on Software Engineering*, 24(8), pp. 629-639, 1998.

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

[8] N. I. Churcher and M. J. Shepperd, "Comments on 'A Metrics Suite for Object-Oriented Design'", *IEEE Transactions on Software Engineering*, 21(3), pp. 263-5, 1995.
[9] M. Davidsson, Zheng, J., Nagappan, N., Williams, L., Vouk, M., "GERT: An Empirical Reliability Estimation and Testing Feedback Tool", Proceedings of International Symposium on Software Reliability Engineering, St. Malo, France, pp. 269-280, 2004.
[10] 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.
[11] G. Denaro, Pezze., M, "An empirical evaluation of fault-proneness models", Proceedings of International Conference on Software Engineering, pp. 241-251, 2002.
[12] K. El Emam, "A Methodology for Validating Software Product Metrics," National Research Council of Canada, Ottawa, Ontario, Canada NCR/ERC-1076, June 2000 June 2000.
[13] K. El Emam, S. Benlarbi, and N. Goel, "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics", *IEEE Transactions in Software Engineering*, 27(7), pp. 630-650, 1999.
[14] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*: Brooks/Cole, 1998.
[15] IEEE, "IEEE Standard 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology," 1990.
[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] H. F. Kaiser, "An Index of Factorial Simplicity", *Psychometrika*, 39(pp. 31-36, 1974.
[19] S. Kan, *Metrics and Models in Software Quality Engineering*: Addison Wesley, 2003.
[20] T. M. Khoshgoftaar and J. C. Munson, "Predicting Software Development Errors using Software Complexity Metrics", *IEEE Journal on Selected Areas in Communications*, 8(2), pp. 253-261, 1990.
[21] T. M. Khoshgoftaar, Munson, J.C., Lanning, D.L., "A comparative study of Predictive Models for Program Changes During System Testing and Maintenence", Proceedings of International Conference on Software Maintenance, pp. 72-79, 1993.
[22] T. M. S. Khoshgoftaar, N., "Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques", *Empirical Software Engineering*, 8(3), pp. 255-283, 2003.
[23] D. G. Kleinbaum, Kupper, L.L., Muller, K.E., *Applied Regression Analysis and Other Multivariable Methods*. Boston: PWS-KENT Publishing Company, 1987.
[24] T. J. McCabe, "A Complexity Measure", *IEEE Transactions on Software Engineering*, 2(4), pp. 308-320, 1976.
[25] P. Mohagheghi, Conradi, R., Killi, O.M., Schwarz, H., "An Empirical Study of Software Reuse vs. Reliability and Stability", Proceedings of International Conference on Software Engineering, pp. 282-292, 2004.
[26] N. Nagappan, "A Software Testing and Reliability Early Warning (STREW) Metric Suite," in *Computer Science Department*. PhD Thesis, Raleigh: North Carolina State University, 2005.
[27] N. Nagappan, L. Williams, and M. A. Vouk, "Towards a Metric Suite for Early Software Reliability Assessment", Proceedings of International Symposium on Software Reliability Engineering Fast Abstract, Denver, CO, pp. 2003.
[28] N. Nagappan, Williams, L., Osborne, J., Vouk, M., Abrahamsson, P., "Providing Test Quality Feedback Using Static Source Code and Automatic Test Suite Metrics", Proceedings of International Symposium on Software Reliability Engineering, Chicago, IL, pp. 2005.
[29] N. Nagappan, Williams, L., Vouk, M., Osborne, J., "Early Estimation of Software Quality Using In-Process Testing Metrics: A Controlled Case Study", Proceedings of Third Software Quality Workshop, pp. 46-52, 2005.
[30] N. Nagappan, Williams, L., Vouk, M., Osborne, J., "Using In-Process Testing Metrics to Estimate Software Reliability: A Feasibility Study", Proceedings of Fast Abstract: International Sysmposium on Software Reliability Engineering, pp. 21-22, 2004.
[31] D. S. Rosenblum, "A practical approach to programming with assertions", *IEEE Transactions on Software Engineering*, 21(1), pp. 19-31, 1995.
[32] R. Subramanyam and M. S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects", *IEEE Transactions on Software Engineering*, 29(4), pp. 297-310, 2003.
[33] 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.
[34] J. Troster, "Assessing Design-Quality Metrics on Legacy Software," Software Engineering Process Group, IBM Canada Ltd. Laboratory, Ontario 1992.
[35] M. Vouk, Tai, K-C., "Multi=Phase Coverage and Risk Based Software Reliability Modeling", Proceedings of CASCON, pp. 513-523, 1993.