Using In-Process Testing Metrics to Estimate Software Reliability: A Feasibility Study

Nachiappan Nagappan[1], Laurie Williams[1], Mladen Vouk[1], Jason Osborne[2]
[1] *Department of Computer Science, North Carolina State University, Raleigh, NC 27695*
[2] *Department of Statistics, North Carolina State University, Raleigh, NC 27695*
{nnagapp, lawilli3, vouk, jaosborn}@ncsu.edu

Abstract

In industry, information on field reliability of a product becomes available too late in the software development process to affordably guide any corrective actions. A step towards remediation of this problem lies in the ability to provide an early estimation of software reliability. This paper presents a suite of in-process metrics that leverages the software testing effort to provide (1) an assessment of potential software field reliability in early software development phases and (2) the identification of software elements that may be fault-prone. A structured feasibility study conducted at North Carolina State University motivated further research.

1. Introduction

In industry, estimates of software field reliability are often available too late to affordably guide corrective actions to the quality of the software. True field reliability cannot be measured before a product has been completed and delivered to an internal or external customer. Field reliability is calculated using the number of failures found by these customers. Because this information is available late in the product lifecycle, corrective actions tend to be expensive [2]. Software developers can benefit from an early warning regarding the quality of their product.

We suggest that this early warning be built from a collection of internal metrics that are correlated with reliability, an external measure. An internal metric, such as the cylomatic complexity [8], is a measure derived from the product itself [6]. An external measure is a measure of a product derived from the external assessment of the behavior of the system [6]. For example, the number of defects found in system test is an external measure.

Software reliability is defined as the probability that the software will work without failure under specified conditions and for a specified period of time [9]. *Our research **objective** is to construct and validate a set of easy-to-measure, in-process metrics that a) provides an early indication of an external measure of system reliability and b) can be used to identify fault-prone programs (or) modules.* To this end, we have created a metric suite we call the Software Testing and Reliability Early Warning metric suite for Java (STREW-J). The metric suite is applicable for development teams that write extensive automated test cases. In this paper, we present the results of a feasibility study designed to analyze the capabilities of the STREW metrics suite. The study was run with junior and senior students in a software engineering class at North Carolina State University (NCSU).

Structural object-orientation (O-O) measurements, such as those in the CK metric suite [4], have been used to evaluate and predict fault-proneness [1, 3]. These metrics can be useful early indicators of product quality [1].

2. STREW - J Metric Suite

Currently, the STREW-J Version 1.3 consists of the seven candidate metrics. The metrics are intended to cross-check each other and to triangulate upon a reliability estimate. 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. The seven metrics are now described:

- *number of test cases/source lines of code (R1)* is an indication of whether there are too few test cases written to test the body of source code;
- *number of test cases/number of requirements (R2)* is an indication of the thoroughness of testing relative to the requirements;
- *test lines of code/source lines of code (R3)* is an indication of whether the ratio R1 was misleading as there might have been fewer, but more comprehensive, test cases;
- *number of assertions/source lines of code (R4)* serves as a crosscheck for both R1 and R2 because there are few test cases but each have a large number of assertions ;
- *number of test classes/number of source classes (R5)* is an indication of how thoroughly the source classes have been tested;
- *number of conditionals (for, while, if, do-while, switch)/ number of source lines of code (R6)* is an indication of the extent to which conditionals are used. The use of conditional statements increases the amount of testing required because there are more logic and data flow paths to be verified [7];

- *number of lines of code/number of classes (R7)* is used as an indicator for estimating reliability when prior releases of the software are available so that any change in the relative class size can be analyzed to determine if the increase in the size of the class has be accompanied by a corresponding increase in the testing effort [5].

3. Structured Experiment

A multiple linear regression model was built based on Java programs written by students over a three-week period. The programs implemented a simplistic inventory system. The students wrote unit test cases using the JUnit[1] testing framework. The assignment stated that 100% statement coverage was required. Thirty one working programs with test cases were used for the analysis. The size of the inventory programs ranged in source lines of code (Mean=347.4, Std. Dev.=91.6) and test lines of code (Mean=209.8, Std. Dev.=145.1). These 31 programs were tested against a set of six black box test cases to estimate the actual reliability of the system using the Nelson model.

Using the technique of data splitting a random sample of 21 programs was selected to build the regression model, and the remaining 10 programs were used to verify the prediction accuracy. The regression equation built had an R^2 value of 0.405, (F=1.262, p=0.340) and is shown below in Equation 1.

Reliability Estimate = 1.114 - 0.0842*R1 - 0.0118*R2 - 0.0247*R3 + 1.954*R4 - 0.0036*R5 – 2.301*R6 - 0.0022*R7 **(1)**

Figure 1 displays the estimated and the actual reliability (lower and upper confidence bounds are given by LCB and UCB); the estimated values are very close to the estimated values. This result demonstrates the potential of using in-process testing metrics to predict software reliability early in the development process.

Further, we performed a discriminant analysis to identify fault-prone programs based on the metric suite elements. The fault-prone and not fault-prone programs were classified based on the normal lower bound[2] calculation of system reliability. The discriminant analysis (eigen value=0.481) for the given data was successful in correctly classifying 74.2% (23/31 of the programs. Fourteen of the 16 fault-prone programs were correctly classified, i.e. 87.5% (14/16).

4. Conclusions and Future Work

The main observations of the study are the following:

- A multiple regression approach for empirical reliability estimation using the STREW metric suite is a practical approach to measuring software reliability; and

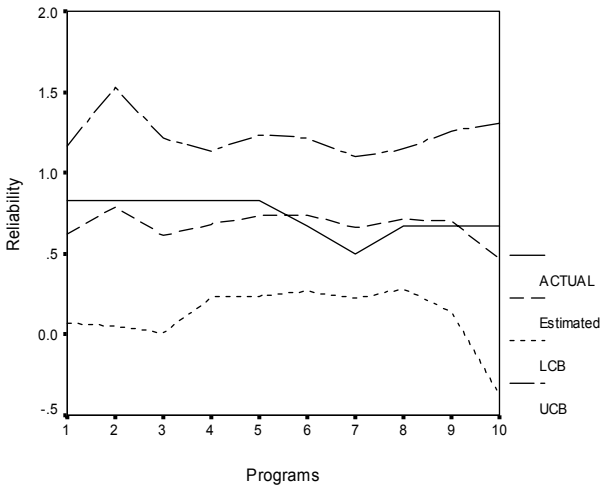

Figure 1: Actual vs. Estimated Reliability

- STREW-based discriminant analysis is a feasible technique for detecting fault-prone programs;

Further industrial case studies are in progress that will assess of the efficacy of the STREW metric. We also plan to update the composition of metric suite to identify the best set of in-process testing as predictors of system reliability. Also we will use the industrial case studies to create standard model parameters which will be of use to developers who are developing new projects without prior comparable historical projects.

Acknowledgements

This work is supported in part by an IBM-Eclipse Innovation Award and by the National Science Foundation under CAREER Grant No. 0346903.

References

[1] V. Basili, Briand, L., Melo, W., "A Validation of Object Oriented Design Metrics as Quality Indicators," *IEEE Transactions on Software Engineering*, vol. Vol. 22, no., 1996.

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

[3] L. C. Briand, Wuest, J., Ikonomovski, S., Lounis, H.,, "Investigation of Object-Oriented Designs:An Industrial Case Study," ICSE 1999, pp.345-354, 1999.

[4] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, 1994.

[5] K. El Emam, Benlarbi, S., Goel, N., Rai, S.N., "The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics," *IEEE Transactions on Software Engineering*, vol. Vol. 27, no., 2001.

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

[7] T. M. Khoshgoftaar, Munson, J.C.,, "Predicting software development errors using software complexity metrics," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 2, pp. 253-261, 1990.

[8] T. J. McCabe, "A Complexity Measure," *IEEE Transactions on Software Engineering*, vol. 2, no. 4, pp. 308-320, 1976.

[9] J. Musa, *Software Reliability Engineering*: McGrw-Hill, 1998.

[1] http://www.junit.org

[2] Lower bound (Reliability) = μ_x-(1.96*Standard deviation of metric Rx)/$\sqrt{n}$. If the reliability is less than the lower bound then it is classified fault-prone, otherwise not fault-prone.