# Using In-Process Metrics to Predict Defect Density in Haskell Programs

Mark Sherriff, Laurie Williams, Mladen Vouk
*Department of Computer Science*
*North Carolina State University, Raleigh, NC 27695*
*{mssherri, lawilli3, vouk}@ncsu.edu*

## Abstract

*In late-stage phases of development, action to correct defects can be cost prohibitive. Effective, efficient, and expressive measures of reliability during the development cycle could aid developers by providing early warning signs of where the system might require modification or further testing. To this end, this paper presents a method for estimating defect density in a system using a suite of internal metrics for Haskell programs. A feasibility study of this method was conducted by analyzing the source code of seven released versions of the Glasgow Haskell Compiler. Further studies are being conducted to refine the metric suite and to examine the potential of the method.*

## 1. Introduction

Some profess that functional languages offer a good balance between productivity and reliability, maintainability, and efficiency [1]. Recently, research and projects have been undertaken to take advantage of these benefits [3], specifically of the Haskell language. Through our research, we aim to add to the body of reliability knowledge for systems built with the Haskell functional programming language. *Our research objective is to construct and validate an easy-to-measure, internal, in-process method that can be used as an early indication of an external measure of defect density.*

Our proposed method uses a suite of internal, in-process metrics to estimate defect density in a Haskell program. We call the suite of metrics the Software Testing and Reliability Early Warning for Haskell (STREW-H). The candidate metrics selected for initial consideration in STREW-H range from testing metrics to structural metrics to compiler warnings. A feasibility study using a subset of the STREW-H metrics to estimate defect density was performed using metrics from seven versions of the open source Glasgow Haskell Compiler source code. We believe our methods will aid engineers by providing an early warning as to the defect density that might be within their system.

## 2. STREW-H

Nagappan et al. are researching the early estimation of reliability growth of Java programs using in-process testing metrics. This early estimation provides feedback to developers so that they can correct faults during the development process and can increase the testing effort, if necessary, to provide added confidence in the software. Nagappan's method, the Software Testing Reliability Early Warning for Java systems (STREW-J) [2, 5], uses a suite of metrics that can be automatically gathered and can be used to provide color-coded feedback to programmers on the reliability of various parts of their system and the thoroughness of their test effort. While the STREW-J metric suite is still in development, early results from a feasibility study and a structured experiment [2, 5] have shown that a regression equation can be formed to provide a practical estimate of software reliability.

Based on Nagappan's work [5], we propose the STREW-H metric suite. Because of the differences in the language paradigms, some of the metrics are not as applicable with functional languages. For example, lines of code [4] is a commonly used metric that is easy to gather with either paradigm, while metrics related to class structures are not as relevant in a functional programming environment.

We utilized the STREW-J metric suite as a starting point for the STREW-H. We eliminated the metrics that were not applicable for functional languages and made additions based upon a review of the literature and upon expert opinion. Expert opinion was gathered via interviews with 12 Haskell researchers at Galois Connections, Inc.[1] and with members of the Programatica team[2] at The OGI School of Science & Engineering at OHSU (OGI/OHSU). Research was performed to validate the inclusion of these potential metrics in the STREW-H. From these sources, we propose an initial set of metrics for the STREW-H version 0.1, as follows:

- *number of test case asserts / source lines of code*
- *IO monadic lines of code / source lines of code*
- *number of type signatures / source lines of code*
- *number of overlapping patterns / source lines of code*
- *number of duplicate exports / source lines of code*
- *number of missing fields / source lines of code*
- *number of missing methods / source lines of code*
- *number of incomplete patterns / source lines of code*
- *number of missing signatures / source lines of code*
- *number of name shadowing / source lines of code*
- *number of unused binds / source lines of code*
- *number of unused imports / source lines of code*
- *number of unused matches / source lines of code*
- *number of test cases / number of requirements*
- *test lines of code / source lines of code*

Through validation with multiple industrial projects, we will refine the proposed metric suite by adding and deleting metrics until we feel we have the minimal set of metrics needed to accurately predict and explain product defect density.

[1] http://www.galois.com/
[2] http://www.cse.ogi.edu/PacSoft/projects/programatica/

## 3. Feasibility Study

A feasibility study was conducted to analyze the potential of a using the STREW-H metric suite to estimate defect density. While our ultimate objective is to use these metrics to estimate reliability, actual reliability data was not available for our feasibility study; defect density data was available. A subset of the metrics from the STREW-H were analyzed. The open source Glasgow Haskell Compiler[3] (GHC) was chosen as the initial test system, since there were seven versions available, along with detailed documentation and defect logs.

Four metrics were chosen for the feasibility study. These metrics were chosen based on expert recommendation and their relation to the STREW-J metric suite. The metrics include:

- Monadic Code Instances / Source KLOC (T1)
- Test Cases / Source KLOC (T2)
- Test LOC / Source KLOC (T3)
- Type Signatures / Source KLOC (T4)

A multiple regression analysis[4] was performed on these four metrics to determine if they were indicative of the number of defects that were found for each version. Six of the seven versions of GHC were randomly chosen to formulate the coefficients in the regression model, Equation 1. Equation 1 was used to predict the defect density of the seventh version. The regression equation formed from the six versions was found to be:

$$\textit{Defect Density (Defects/KSLOC)} = .08 + .0113 * (T1) + .0002 * (T2) + .607 * (T3) - .0762 * (T4) \qquad \textbf{(1)}$$

Figure 1 shows a plot of the actual defect densities with the regression equation used in the model. Using Equation 1, the estimated defect density for the seventh version was 0.04 defects/KLOC, while the actual defect density was 0.07 defects/KLOC. This equation was shown to provide a good estimate of defect density in other versions of the system. While there were not enough test cases to denote statistical significance, the initial results of the study indicate that this method may be an efficient indicator of the defect density.

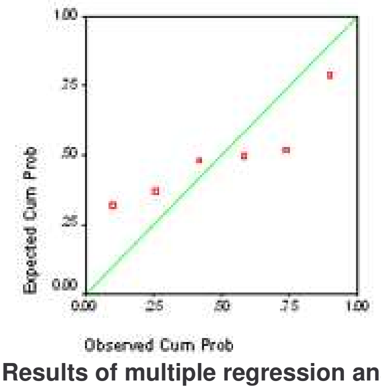

**Figure 1. Results of multiple regression analysis.**

**Table 1. Collected data from seven versions of GHC.**

| Ver. | Source KLOC | Mon. Inst. | Test Cases | Test LOC | Type Sigs. | Defect densiy |
|---|---|---|---|---|---|---|
| 4.08 | 99.73 | 1023 | 226 | 1444 | 11737 | 0.49 |
| 5.00.2 | 140.67 | 2789 | 0 | 0 | 16739 | 0.33 |
| 5.02.2 | 157.84 | 2526 | 0 | 0 | 22338 | 0.29 |
| 5.04 | 211.42 | 6453 | 76 | 1749 | 28678 | 0.04 |
| 5.04.3 | 205.95 | 6453 | 76 | 1749 | 27991 | 0.08 |
| 6.0 | 213.04 | 6892 | 76 | 1749 | 28179 | 0.08 |
| 6.01 | 216.13 | 7258 | 76 | 1749 | 28600 | 0.07 |

A limitation of the STREW-H method of estimating defect density is that it is not based on any operational profile of the system. While utilizing operational profiles to estimate system defect density would be beneficial, it is often cost and time prohibitive. Rivers and Vouk [6] have shown that non-operational testing is related to field quality, and thus there is potentially value in this method.

## 4. Conclusions and Future Work

Having an early warning system to estimate defect density would aid developers by giving them an indication as to potential problems in the system. We can leverage metrics that are readily available in any system to help provide this defect density estimate. A method for estimating the defect density of software written in a functional language environment has been presented in this paper. The method utilizes in-process metrics to estimate defect density. An automated tool is currently being created to automatically gather this information and provide it to developers while they are still implementing code and can affordably make corrective actions. An initial feasibility was performed using a subset of metrics from the STREW-H. Results motivate further study.

## Acknowledgements

This work is supported by the National Science Foundation.

## References

[1] Breazu-Tannen, V., Buneman, O. P., and Gunter, C. A. "Typed functional programming for rapid development of reliable software." In J. E. Gaffney, editor, Productivity: 18 Progress, Prospects, and Payoff. June, 1988. pp. 115-125.

[2] Davidsson, M., Zheng, J., Nagappan, N., Williams, L., and Vouk, M., "GERT: An Empirical Reliability Estimation and Testing Feedback Tool," International Symposium on Software Reliability Engineering, 2004, To appear.

[3] Halgren, Thomas, "Tools from the Programatica Project," presented at ACM SIGPLAN Haskell Workshop, 2003.

[4] Khoshgoftaar, T. and Munson, J., "The Lines of Code Metric as a Predictor of Program Faults: A Critical Analysis," Computer Software and Applications Conference, 1990, pp. 408-413.

[5] N. Nagappan, Williams, L., Vouk, M.A., "Towards a Metric Suite for Early Software Reliability Assessment," International Symposium on Software Reliability Engineering, FastAbstract, Denver,CO, pp.238-239, 2003.

[6] Rivers A. and Vouk, M.A., "Resource Constrained Non-Operational Testing of Software," Proceedings of ISSRE 98, 9th International Symposium on Software Reliability Engineering, Paderborn, Germany, 4-7 Nov., 1998.

[3] http://www.haskell.org/ghc/

[4] SPSS was used for to compute the regression equation.