"Good Enough" Software Reliability Estimation Plug-in for Eclipse

Nachiappan Nagappan, Laurie Williams, Mladen Vouk
Department of Computer Science
North Carolina State University, Raleigh 27695
{nnagapp, lawilli3, vouk} @unity.ncsu.edu

Abstract

Programmers who use the test-driven development practice of the Extreme Programming methodology write extensive automated unit and acceptance tests. This paper describes an Eclipse plug-in that utilizes the results of this automated testing, and in combination with a suite of internal product metrics, provides an early assessment of product reliability. We discuss the correlation between the metrics we use and the reliability of the developed software, as well as the general functional characteristics of the alpha version of our Eclipse plug-in.

1 Introduction

In test-driven development (TDD) [3] practice of the Extreme Programming (XP) [2] software development methodology, programmers are encouraged to write **automated** unit and acceptance test cases throughout the software development cycle. This provides a powerful verification and validation focus that, when properly implemented, can result in considerable quality benefits. For example, in our studies, we met an industrial team which, when they started using TDD, found that the practice provided about 40% decrease in the number of "functional verification test" defects as compared to a similar legacy project [15, 22]. The Eclipse plug-in tool we developed, further leverages the benefits of these extensive automated test cases by providing feedback on the quality of the testing process and on its possible impact on the final quality of the software

2 Software Metrics

An internal metric, such as the number of lines of code, is a measure derived from the product itself [14]. An external metric is a measure of a product derived from assessment of its behavior [14]. For example, the number of defects found during testing is an external measure. Internal metrics also include object-oriented measurements, such as those defined in the Chidamber-Kemerer (CK) [6] and MOOD [5] object-oriented (OO) metric suites. The ISO/IEC standard [14] states that "Internal metrics are of little value unless there is evidence that they are related to external quality." *Objective of this work is to create and validate an Eclipse-based tool set based on easy-to-measure internal metrics that would provide feedback to TDD practitioners on the quality of their effort and product.* This work is building on that reported in [13, 20, 21] and it illustrates how OO and other metrics can be used to provide early quality predictions.

There is a growing body of empirical evidence that validates higher-order metrics, such as OO metrics [1, 4] as early predictors of the field quality of software. Validation of the metric relevance requires 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 of the software in a statistically significant and stable way [9].

The Eclipse plug-in we developed supports the collection and analysis of an initial set of internal metric data, and calculation of a reliability estimate based on these measures. The initial metrics suite includes

- *number of test cases/source lines of code (R1)* ;
- *number of test cases/number of requirements (R2)*;
- *test lines of code/source lines of code (R3)*;
- *number of asserts/source lines of code (R4)*. This metric serves as a control for both R1 and R2. For example, in the case when there are few test cases but each has a large number of successful calls to the source program, the

metric suite may penalize the developer. R4 helps mitigate this;
- *code coverage (C)* which measures extent to which the source program is covered by testing and also is an indicator of the accuracy of R3.

The work of Vouk and Tai [21] has shown that similar structural metrics and ratios (e.g., number of lines of code changed in a product, or number of test cases per changed line of code) may exhibit strong correlation with field quality of the product. The validity of the above described metric suite is still under investigation, but initial results are encouraging. We plan to add/delete metrics to the current suite and update the plug-in based on the results of further validation.

3 Reliability model

TDD software engineers rapidly iterate through the following:

1. Writing a small number of **automated** test cases;
2. Running these unit test cases to ensure they fail (they must fail, since there is no code yet this test-case software is intended to test – there are only stubs);
3. Implementing the code which should allow the unit test cases to pass;
4. Re-running the unit test cases to ensure they now pass with the developed code – and fixing the problems if they fail to pass;
5. Refactoring [10] of the implementation and he test code, as necessary; and
6. Periodically re-running all the test cases in the code base to ensure that the new code does not break any previously-running test cases.

In step 2, failures are a normal part of the process. At the end of step 4, however, every automated test case should pass. This TDD sequence presents challenges for reliability estimation because (1) failures at step 2 are not an indication of poor reliability and (2) by end of step 4, there are "no failures" based upon the set of automated test cases that have been written. Because of the lack of failures by end of step 4, reliability models based on failure rates may not be suitable for reliability estimation.

A number of software reliability growth models are based on the non-homogenous Poisson Processes (NHPP) [23] assumptions. These models, such as the Musa [17, 18] and the Goel-Okumoto[11, 18] models, require measurement of failure intensity to work. Since in TDD, there are nominally "no failures," reliability models based on failure rates have a problem. Instead, an appropriate "no failure" estimation model needs to be used. Some are described in [7, 8, 16]**.** Our plug-in currently uses reliability estimation methods based on the "no failure" approach described in [8, 12, 16], . It basically says that, given N representative test cases that do not fail, one can compute a meaningful upper confidence bound on the actual system failure rate Θ that states that this rate is below some level θ [12] with probability $(1-\alpha)$. The relationship between these factors is given by $1- (1-\theta)^N <= \alpha$ [19]. This θ value is then a lower confidence bound on the failure rate of our software. While we assume that TDD test cases are representative of the operational profile of software, we note that we continue to study this approach in the context of the impact of the degree of to which TDD test-cases are representative of the actual operational profile (field executions).

4 Eclipse Plug-In

Most TDD test cases tend to be written using xUnit[1] or Framework for Integrated Test (FIT)[2] automated testing tools. Our alpha version of the Eclipse plug-in uses these tests to estimate reliability and to provide feedback on the testing effort. The plug-in contains three main options.

- Read File: This option reads through the source and the test code and gathers data for the metrics suite.
- Zero-Failure model: This model computes the reliability of the software based on mechanisms discussed in Section 3. Visual feedback is provided to show the developer the current reliability status.
- Empirical model: This option provides color-coded, visual feedback on the quality of the testing effort based upon the metrics suite, as shown in Figure 1. The color coding provides feedback on the metrics relative to historical data from comparable projects; data on comparable projects is provided by the tool

[1] http://xprogramming.com/software.htm
[2] http://fit.c2.com/

user. Acronyms are : LL = Lower Limit; μ_x, = Mean of metric x, R_x is the metric id, and

$LL(R_x) = \mu_x - 1.96$[3]*Standard deviation of metric R_x/sqrt(n) – assumes normal distribution of errors

RED: $R_x < LL$ (*Metric*)
ORANGE: LL (*Metric*) $\leq R_x \leq \mu_x$
GREEN: $R_x > \mu_x$

We have performed an initial feasibility study of this tool on 13 programs developed in a senior-level software engineering course at North Carolina State University (NCSU). The feasibility study demonstrated a positive correlation among relevant ratios indicating that they are not giving conflicting information. The results of a bivariate analysis are shown in Table 1. It indicates that there exists a statistically significant[4] relationship between the ratios *R1, R2* and the reliability estimate. But there is no statistically significant relationship in the case of R3 and R4 as they depend heavily on developer characteristics. Furthermore, we feel that in this case the relationship between the code coverage and the estimated reliability is not relevant because the students were specifically assigned to create test cases with high code coverage, and that might not be representative of industrial software development practices.

Table 1: Pearson correlation and statistical significance results between the ratios and the reliability estimate

θ	R1	R2	R3	R4	C
.01	0.629, p<.021	0.992, p<.000	0.302, p<.316	0.469, p<.106	0.118, p<.702
.05	0.703, p<.007	0.814, p<.001	0.393, p<.184	0.541, p<.056	0.039, p<.900
.10	0.616, p<.025	0.584, p<.036	0.414, p<.159	0.502, p<.081	-0.07, p<.817

The LL and mean value calculations are illustrated for the four ratios in Table 2. We used these values to provide demo feedback on a bowling game simulation program written by one of the authors. Output is shown in Figure 1.

[3] 1.96 is the t-table value at 95% confidence interval. If sample size is greater than 30 then use the normal table at the desired confidence interval.
[4] Statistical Significance is determined at (p<0.05)

Table 2: Color coded feedback

Ratio	LL	Mean	"Bowling game" metrics	Color
R1	0.0358	0.0476	0.045	Orange
R2	1.1252	1.8192	0.550	Red
R3	0.3133	0.4433	0.870	Green
R4	0.0718	0.1085	0.045	Red

Our plug-in requires the program to include the "gert" package we developed earlier. There are several parameters in the package that can be controlled by passing appropriate values in the JUnit test program to the reliability tool as illustrated below in Program 1.

- setConfidence(float): This parameter is used to fix the desired confidence level. The default value is 0.95
- setStatlevels(float): This option allows the user to modify the LL prediction of the metric. By default this is set at 1.96 at 95% confidence. Users may find it appropriate to increase or decrease the statistical confidence level. The student-t table values are used when the number of samples is less than 30; otherwise a standard normal distribution value is assumed at the appropriate confidence level.
- setEmpPar(double,double,double,double,double): This function as illustrated in Program 1 below. It allows the developers to pass parameters collected form previous projects that have appropriate quality to serve as guidelines for the current project. This forms the basis of our empirical model. The ratios, the LL, and the means are calculated automatically by the system.

Program 1: Sample Junit program running "gert" package

```
import junit.framework.*;
import java.io.*;
import gert.*;

public class xpBowlTest extends
TestCase
{
  public xpbowling x;
  static gert.Test test1
          = new gert.Test();
```

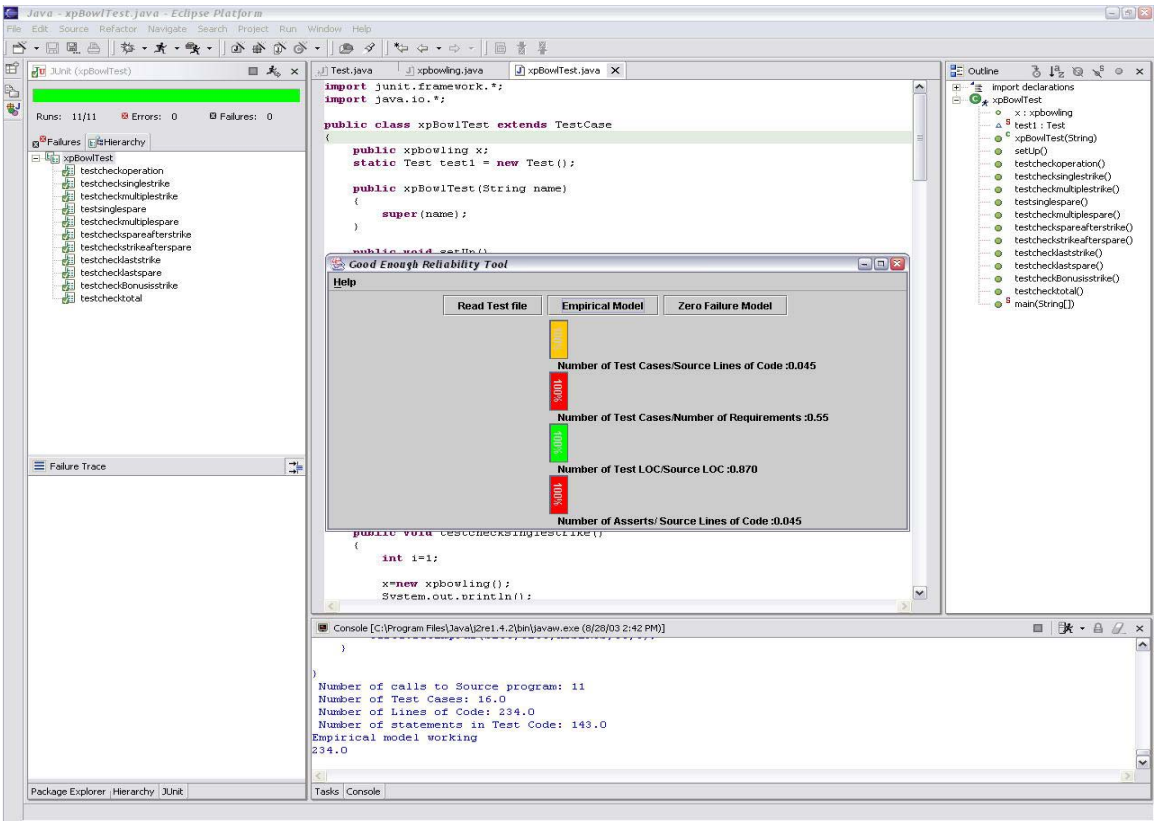

Figure 1: Reliability tool support-Color coded feedback

```
      JUNIT TESTS GO HERE…..

  public static void main(String[]
args)
  {
      double[] SLOC= {..};
      double[] TLOC= {..};
            double[] ASSERTS={..};
      double[] TC={..};
      double[] C={..};
      test1.setConfidence(0.90);
      test1.setStatlevels(0.99);
      test1.setEmpPar(SLOC,TLOC,TC
,SLOC,C);

  }
```

5. Conclusion

We have developed an Eclipse plug-in that allows TDD practitioners to compute an early estimate of the quality of the software their testing may be producing. The plug-in is an alpha version still under assessment. Initial results of this assessment are encouraging. We have described the basics of the methodology and assumptions behind the plug-in methods, and have illustrated demonstration output from the tool..

However, metrics and model validation will require extensive analysis in a multitude of environments. We plan an extensive assessment of our metric suite in a variety of different industrial and academic environments. Through this work, we will refine the metric suite and our plug-in by adding and deleting metrics based on the results of these case studies. Further we plan to develop standards for these metrics based on data from industry and academia so that it may not be required for developers to always calibrate on their own projects before receiving feedback on the quality of a testing effort.

Acknowledgements

This research was funded by IBM via an Eclipse Innovation Award.

About the Authors

Nachiappan Nagappan is a Doctoral Student in the Computer Science department at NCSU. Nachiappan holds a B.Tech degree from the University of Madras and a Masters from NCSU. **Dr. Laurie Williams** is an Assistant Professor in the Department of Computer Science at NCSU. She obtained her PhD from the University of Utah. **Dr. Mladen Vouk** received Ph.D. from the King's College, University of London, U.K. He is Associate Vice-Provost for Information Technology and a Professor of Computer Science at N.C. State University. He is also the Technical Director of the N.C. State Center for Advanced Computing and Communication.

References

1. Basili, V., L. Briand, and W. Melo, *A Validation of Object Oriented Design Metrics as Quality Indicators.* IEEE Transactions on Software Engineering, 1996. **22**(10): p. 751-761.
2. Beck, K., *Extreme Programming Explained, Embrace Change*. 2000: Addison Wesley.
3. Beck, K., *Test Driven Development- by Example*. 2003, Boston: Addison-Wesley.
4. Briand, L., K.E. Emam, and S. Morasca, *Theoretical and Empirical Validation of Software Metrics*, in *ISERN Technical Report 95-03*. 1995.
5. Brito e Abreu, F. *The MOOD Metrics Set*. in *ECOOP '95 Workshop on Metrics*. 1995.
6. Chidamber, S.R. and C.F. Kemerer, *A Metrics Suite for Object Oriented Design*, in *IEEE Transactions on Software Engineering*. 1994.
7. Duane, J.T., *Learning Curve Approach to Reliability Monitoring.* IEEE Transactions on Aerospace, 1964.
8. Ehrenberger, W., *Statistical Testing of Real-Time Software*, in *Verification and Validation of Real-Time Software*, W. Quirk, J., Editor. 1985, Springer-Verlag: New York.
9. El Emam, K., *A Methodology for Validating Software Product Metrics*. June 2000, National Research Council of Canada: Ottawa, Ontario, Canada.
10. Fowler, M., et al., *Refactoring: Improving the Design of Existing Code*. 1999, Addison Wesley: Reading, Massachusetts.
11. Goel, A., Okumoto, K., *Time-Dependant Error-Detection Rate Model for Software Reliability and other Performance Measures.* IEEE Transactions on Reliability, 1979: p. 206-211.
12. Hamlet, D., Voas J. *Faults on Its Sleeve: Amplifying Software Reliability Testing*. in *International Symposium on Software Testing and Analysis*. 1993. Cambridge, MA.
13. Harrison, R., S.J. Counsell, and R.V. Nithi, *An Evaluation of the MOOD Set of Object-Oriented Software Metrics*, in *IEEE Transactions on Software Engineering*. June 1998. p. 491-496.
14. ISO/IEC, *DIS 14598-1 Information Technology - Software Product Evaluation*. 1996.
15. Maximilien, E., Williams, L. *Assessing Test-Driven Development at IBM*. in *International Conference on Software Engineering*. 2003. Portland, OR.
16. Miller, K.W., Morell, L.J., Noonan, R.E., Park, S.K., Nicol, D.M., Murrill, B.W., Voas, J.M., *Estimating the Probability of Failure When Testing Reveals no Failures.* IEEE transactions on Software Engineering, 1992.
17. Musa, J., *Theory of Software Reliability and its Applications.* IEEE Transactions on Software Engineering, 1975: p. 312-327.
18. Musa, J., Ianino, A.,Okumoto, K., *Software Reliability:Measurement, Prediction, Application*. 1987, New York: McGraw Hill.
19. Thayer, R., Lipow, M., Nelson, E., *Software Reliability*. 1978, Amsterdam: North-Holland.
20. Vouk, M.A., Jones, W., *Software Reliability Field Data Analysis*, in *Handbook of Software Reliability Engineering*, M. Lyu, Editor. 1996, McGraw Hill.
21. Vouk, M.A., Tai, K.C. *Multi-Phase Coverage- and Risk-Based Software Reliability Modeling*. in *CASCON '93*. 1993.
22. Williams, L., Maximillian, E.M., Vouk, M.A. *Test-Driven Development as a Defect-Reduction Practice.* in *International Symposium on Software Reliability Engineering*. 2003. Denver, CO: IEEE-CS.
23. Yamada, S., Osaki,S., *Software Reliability Growth Modeling:Models and Applications.* IEEE Transactions on Software Engineering, Dec 1985: p. 1431-1437.