International Journal of Advances in Engineering Research http://www.ijaer.com

(IJAER) 2013, Vol. No. 5, Issue No. I, January **ISSN: 2231-5152**

STUDY ON PRACTICAL APPLICATIONS TO REGRESSION TESTING

Sanjay Bhardwaj

ABSTRACT

A key improvement was to use a statistical model, such as a logistic-regression model, to quantify the relationship between test-suite and fault characteristics and a test suite's ability to detect a fault. The statistical model can be used to show which test-suite and fault characteristics have important (statistically significant) effects on fault detection, and the direction and magnitude of those effects. This information can, as Chapter 1 explained, improve the science of testing. It can also, as this chapter shows, improve the practice of testing—specifically, the practice of regression testing.

Regression testing is an important step in the software-maintenance cycle that tests modifications made to previously tested software. An iteration of regression testing is an opportunity to catch changes that break the software. It is also an opportunity to learn how to make the next regression-testing iteration more effective.

Key words: *Regression, fault characteristics, software-maintenance, next regression-testing.*

INTRODUCTION

Software defects also have their technical terms. In common parlance, these terms are often used interchangeably, and even among experts they have different meanings in different communities [50]. In this work, mistakes, faults, and failures are treated as distinct, following one version of the definitions in the IEEE Standard Glossary of Software Engineering Terminology [50]. Here, a mistake is considered to be a flaw in a programmer's mental conception of a program that leads to one or more problems in the software. Each of those problems, as it is manifested in the source code of the software, is called a fault. A fault may cause one or more failures, or incorrect behaviors in the software. This work uses the term defects to refer generally to problems with software—mistakes, faults, or failures.

When researchers study the abilities of testing techniques to detect defects, they usually need to collect or create some defective software on which to try the techniques. They can accomplish this in several ways. They can collect natural defects (sometimes called real defects), which are defects made accidentally by the developers of a software product. If the developers have kept good records through a version-control system and a defect-tracking system, then researchers can isolate and reconstruct the defects that have been fixed over time, although this is a painstaking process. Alternatively, researchers can seed defects into a software product—that is, insert them intentionally. Defects can either be seeded by hand or by automated mutation. A mutation fault is a small, typo-like fault in the code, such as changing a plus sign to a minus sign.

REVIEW OF LITERATURE

To understand fully how evaluations of testing techniques may depend on the defects used, one must be familiar with the typical procedure for such evaluations. Although some analytical approaches have been proposed [23], by far the most common and practical way to evaluate testing techniques continues to be empirical studies. Typically, these studies investigate hypotheses like "Technique A detects more defects than technique B" or "A detects more than zero defects more often than B" [32].

Empirical studies, by their very nature as sample-based evaluations, always face the risk that different samples might lead to different results. An empirical study must select a sample of software to test, a sample of the test suites that can be generated (or recognized) by each technique for the software, and a sample of the defects (typically faults) that may arise in the software. Because different studies usually use different samples, one study might report that technique A detects more faults than B, while another would report just the opposite. More and more, published empirical studies of software testing are acknowledging this as a threat to external validity [5, 26, 53].

MATERIAL AND METHOD

To date, research has paid much attention to the problem of constructing a test suite initially but little attention to improving upon the initial suite's fault detection given some experience. Much research has studied the choice of testing technique and other factors that make for an effective test suite, without necessarily considering regression testing (Section 2.3). Research specifically on regression testing has typically focused on selecting or prioritizing test cases from the initial test suite in order to improve the speed or cost of fault detection [28, 55, 54]. But, so far, no one has used feedback from previous iterations of regression testing to design a test suite that detects more faults, or more important faults, in the current iteration.

Suppose that a team of testers needs to test version n of a software product. For versions 1, . . . , n − 1, the test suites used, the faults found by each test suite, and the faults found after testing (e.g., by users) have been recorded. This section describes three ways that testers could use this information with a statistical model of fault detection to better test version n.

Statistical model (fabricated, not based on real data):

Logit (Pr(Det)) = 0.3T.Events + 3T.Pairs − 10F.CovBef + 40T.Pairs × F.CovBef

Characteristics of test suites used for versions 1, . . . , n − 1:

T.Events	T.Pairs
2.0	0.20

Known faults in version 1:

	F.CovBef = 0.3	F.CovBef = 0.7	Total
Found by GUI testing	30	30	60

Known faults in versions 1, . . . , n − 1 :

	F.CovBef = 0.3	F.CovBef = 0.7	Total
Found by GUI testing	200	200	400
Found by users, etc.	100	400	500

Figure .1: Statistical model and other data used in examples

Three scenarios are described, and each is illustrated with at least one example. To be concrete, the examples use fictitious data. (The next section demonstrates the scenarios with real data.) While the scenarios make no assumptions about the testing technique or statistical model being used, the examples assume that the testers are using GUI testing and a logistic-regression model similar to the ones developed in the previous chapter. The logistic-regression model and other data used in the examples are summarized in Figure 1.

CONCLUSION

It shows how feedback—in the form of test-suite and fault histories from previous iterations of regression testing—together with statistical models of fault detection can help testers detect more, and more important, faults as regression testing progresses. The next section describes three such scenarios. To make the descriptions more palpable, each scenario is illustrated with a simple example based on fictitious feedback and models. While fictitious data, in its contrived neatness, helps to clearly explain the scenarios, real data can better show their strengths and weaknesses. That is why Section 5.2 demonstrates the scenarios with data from two versions of a real application. Section 5.3.1 outlines a more comprehensive evaluation of these scenarios and describes several additional scenarios.

REFERENCES

[1] First International Workshop on Testing Techniques and Experimentation Benchmarks for Event-Driven Software (Apr. 2009).

[2] Agrawal, H., Horgan, J. R., Krauser, E. W., and London, S. Incremental regression testing. In Proceedings of ICSM '93 (Washington, DC, USA, 1993), IEEE Computer Society, pp. 348-357.

[3] Agresti, A. An introduction to categorical data analysis, second ed. John Wiley & Sons, 2007.

[4] Andrews, J. H., Briand, L. C., and Labiche, Y. Is mutation an appropriate tool for testing experiments? In Proceedings of ICSE '05 (2005), pp. 402-411.

[5] Andrews, J. H., Briand, L. C., Labiche, Y., and Namin, A. S. Using mutation analysis for assessing and comparing testing coverage criteria. IEEE Trans. Softw. Eng. 32, 8 (2006), 608-624.

[6] Basili, V. R. Software development: a paradigm for the future. In Proceedings of COMPSAC '89 (1989), IEEE Computer Society, pp. 471-485.

[7] Basili, V. R., and Perricone, B. T. Software errors and complexity: an empirical investigation. Commun. ACM 27, 1 (1984), 42-52.

[8] Basili, V. R., and Selby, R. W. Comparing the effectiveness of software testing strategies. IEEE Trans. Softw. Eng. 13, 12 (1987), 1278-1296.

[9] Basili, V. R., Shull, F., and Lanubile, F. Building knowledge through families of experiments. IEEE Trans. Softw. Eng. 25, 4 (1999), 456-473.

[10] Briand, L. C., Melo, W. L., and Wst, J. Assessing the applicability of fault-proneness models across object-oriented software projects. IEEE Trans. Softw. Eng. 28, 7 (2002), 706-720.

[11] Brooks, F. P. The Mythical Man-Month: Essays on Software Engineering, anniversary ed. Addison-Wesley Pub. Co., 1995.

[12] Cai, K.-Y., Li, Y.-C., and Liu, K. Optimal and adaptive testing for software reliability assessment. Information and Software Technology 46, 15 (2004), 989-1000.

[13] Clark, B., Devnani-Chulani, S., and Boehm, B. Calibrating the CO-COMO II post-architecture model. In Proceedings of ICSE '98 (Washington, DC, USA, 1998), IEEE Computer Society, pp. 477-480.

[14] Clarke, L. A. A system to generate test data and symbolically execute

programs. IEEE Trans. Softw. Eng. 2, 3 (1976), 215-222.

[15] Davis, M. D., and Weyuker, E. J. Pseudo-oracles for non-testable programs. In Proceedings of the ACM '81 conference (New York, NY, USA, 1981), ACM, pp. 254-257.

[16] Do, H., and Rothermel, G. On the use of mutation faults in empirical assessments of test case prioritization techniques. IEEE Trans. Softw. Eng. 32, 9 (2006), 733-752.

[17] Eaton, C., and Memon, A. M. An empirical approach to testing web applications across diverse client platform configurations. International Journal on Web Engineering and Technology, Special Issue on Empirical Studies in Web Engineering 3, 3 (2007), 227-253.

[18] Elbaum, S., Gable, D., and Rothermel, G. Understanding and measuring the sources of variation in the prioritization of regression test suites. In Proceedings of METRICS '01 (Washington, DC, USA, 2001), IEEE Computer Society, pp. 169-179.