Applying a Semantic Fault Model to the Empirical Study of Corrective Maintenance

Jane Huffman Hayes (corresponding author) [††]
Laboratory for Advanced Networking
Computer Science
University of Kentucky
301 Rose Street
Lexington, KY 40506-0495
(+1) 859-257-3171 (859) 323-3740fax
hayes@cs.uky.edu

Jeff Offutt[†]
Information and Software Engineering
George Mason University
ISE Department, 4A4
Fairfax, VA 22030-4444
(+1) 703-993-1654
ofut@ise.gmu.edu

Abstract

A full understanding of the characteristics of faults is crucial to several important research areas in testing and software maintenance. Applicable maintenance research areas include change impact analysis, maintainability, regression testing, and comparative evaluation of maintenance techniques. We explore the fundamental nature of faults by looking at the differences between a syntactic and semantic characterization of faults. We offer definitions of these characteristics and explore the differentiation. We discuss the concept of "size" of program faults. This model is then directly applied to changes in corrective maintenance. The measurement of fault size provides interesting and useful distinctions between the syntactic and semantic characterization of changes.

Eighth IEEE Workshop on Empirical Studies of Software Maintenance, **Montreal, Canada, October 2002.**

1 INTRODUCTION

The empirical study of software maintenance is largely dependent on historical data related to the changes made to software systems. We examine this data to extract trends, to understand underlying aspects of software maintenance, and to test our research hypotheses. Sometimes it is necessary for us to develop changes to facilitate our research, just as testing researchers must often seed faults into programs for evaluating testing techniques. As change data is central to our empirical software maintenance research, we want to ensure that it is representative of real-world changes. We feel that one way to achieve this end is to consider a semantic and syntactic characterization of changes.

[†] *Partially supported by the U.S. National Science Foundation under grant CCR-00-97056.* [††] *Partially supported by NASA under grant NAS2-98028.*

By "change" we mean a program modification, including "real" changes made in response to software problem reports or "artificial" changes made for research purposes.

This paper presents our position on the use of a semantic model of changes for empirical study of corrective software maintenance. Section 2 presents our position, introduces the semantic fault model, introduces a semantic change model, and looks at the quantification of size of a change. Section 3 presents the summary and conclusions.

2 POSITION

We claim that a syntactic and semantic characterization of software faults can facilitate several areas of software maintenance research. The fault characterization can be directly applied to software changes made as part of corrective maintenance. Further, the notion of "size" of a fault may help facilitate the experimental study of corrective maintenance changes. Specifically, change impact analysis, maintainability, and comparative evaluation of maintenance techniques can be enhanced through this model of size.

It is the position of the authors thatchanges to software usually lowers its maintainability. Moreover, the authors hypothesize that semantic size is more important than syntactic size in determining if changes result in less maintainable code. If this can be validated, it provides direction to us as we maintain code. For example, if a large semantic change is being made to a module, extra care should be taken to ensure that the maintainability is not further degraded.

2.1 Semantic Model of Faults and Changes

This paper is concerned with faults that compile but that can cause the program to exhibit incorrect behavior, not with syntactic faults that compilers can detect. Faults can be characterized semantically and syntactically [1]. A fault is the difference between the incorrect program and some correct program. Syntactically, a fault may be localized in one statement or may be textually dispersed into several locations in the program. Similarly, a fault may be repairable in many ways, thus there may be multiple correct versions. This definition is in terms of the syntactic nature of a fault. If a fault occurs naturally, then the syntactic nature of the fault can be described by the number of edit changes needed to correct the program. If the fault is inserted or seeded into the program, the syntactic nature of the fault can be described by the number of edit changes to the program. Examples of syntactic characterizations of faults include using an incorrect variable name or misusing parameters.

A fault can also be characterized semantically. Each program *P* can be viewed as having a specification *S* that defines sets *D*, the input domain, *R*, the output range, and a mapping from *D* to *R* $(D \xrightarrow{S} R)$. The program may compute results on a superset of *D*, or if the input is not in *D*, *P* may produce output that is not in *R* (undefined). A semantic characterization of a fault views the faulty program as containing a computation that produces incorrect output over some subset of the input domain. That is, the mapping of inputs to outputs $(D \xrightarrow{S})R \quad (D \xrightarrow{S} R)$ for some subset of *D* [1].

One can quickly see that this model of faults can be directly applied to changes (or fixes) being made as part of corrective maintenance. A change has a syntactic characterization described by the number of fixes needed to correct the program. A change has a semantic characterization described by the subset of the input domain *D* for which the mapping of inputs to outputs is not correct with respect to the specification *S*.

This characterization gives us a different way to look at faults and changes. It becomes useful when we introduce the notion of size of a fault and size of a change. We define the syntactic size of a fault to be the fewest number of statements or tokens that need to be changed to get a correct program. The syntactic size of a change is defined identically. We define the semantic fault size to be the relative size of the subdomain for *D* for which the mapping output is incorrect. The semantic size of a change is defined identically [1].

To illustrate, let us consider very small faults. A syntactically small fault may have only one token or one statement that is incorrect. A semantically small fault may constitute only one or very few inputs of *D* that cause an incorrect mapping to output. A fault that is syntactically small can result in a fault that is very large semantically. Conversely, a large syntactic fault in *P* may affect only a few inputs. There is some intersection where small semantic faults can be modeled as small syntactic faults, and small syntactic faults can result in small semantic faults. For example, consider the following code fragment:

```
for i: = 1 to n do
   A[i] := A [i] +1;
```

Suppose this program fragment contains the very small syntactic fault that the addition operator should be a subtraction operator. The fault will affect every input to the program and will affect every element of A for every input, thus the fault is semantically large [1]. Table 1 lists other examples of the possible intersections.

Table 1. Fault Size Intersection.

Fault Size Characteristic	Large semantic	Small semantic
Large syntactic	Built a Gaussian random number generator when reading values from a pre-existing file was desired	Built mean calculation of list of numbers instead of median
Small syntactic	Used '>' when '<' was desired	Used '>=' when '>' was desired

These observations and examples hold for changes in corrective maintenance as well.

2.2 Implications

This section considers the implications of the syntactic/semantic characterization of changes for corrective maintenance. There are several central issues. The first deals with whether empirical software maintenance studies have focused on changes that were syntactically small without consideration of semantic size. The second issue is if this characterization of changes can help us estimate and/or measure the maintainability of software. If so, this characterization may also help us comparatively evaluate corrective maintenance techniques.

Empirical Studies

In many software maintenance studies, case studies or experiments using problem reports or program changes/fixes were undertaken to examine a research area. For example, De Lucia, Persico, Pompella, and Stefanucci looked at two projects to improve the corrective maintenance cost prediction model currently used in a software company [2]. They looked at the number of maintenance tasks in three categories: maintenance tasks that require source code modification, maintenance tasks requiring fixing of data misalignments through database queries, and maintenance tasks not falling into the above two categories. Their study did not look at the size of the maintenance tasks or the characteristics of the tasks. Bianchi, Caivano, Lanubile, Rago, and Visaggio [3] use faults as a measure of reliability when comparing distributed and collocated projects. They observe that a gap between estimated and actual staffing levels is attributable to a higher number of faults for a Work Packet (a work packet is a functional area of a software system, including programs, library elements, or JCL procedures). The faults are not characterized semantically or syntactically. In both cases above, one can imagine that the semantic size and/or syntactic size of the changes could impact the results. For example, for Bianchi et al. [3] it could be the case that it is not a higher number of faults that caused the result but rather a higher number of <u>small</u> semantic faults. Similarly, one can imagine that tasks may require different levels of effort (and hence cost) based on the semantic and syntactic size of the changes.

It may be the case that there is knowledge to be gained about building maintainable software by looking at the frequency of occurrences of faults by size based on the types of changes made. That is, do we tend to introduce other small semantic faults when we are making changes that are small semantically? What types of faults occur most frequently? If we can collect data on this, we can ensure that we use more operationally representative fault data in our testing research and more realistic change data in our maintenance research.

Measuring Maintainability

There is no clear agreement on how to measure maintainability [8]. Welker suggests measuring it statically by using a Maintainability Index (MI) [4]. A program's maintainability is calculated using a combination of widely used and commonly available measures to form the MI. A large MI value indicates that the program is easy to maintain. The basic MI of a set of programs is a polynomial of the following form:

$$MI = 171 - 5.2*\ln(aveV) - 0.23*aveV(g') - 16.2*\ln(aveLOC) - 50*\sin(\mathrm{sqrt}(2.4perCM))$$

The coefficients are derived from actual usage, with the terms defined as follows:

aveV = average Halstead Volume V per module
aveV(g') = average extended cyclomatic complexity per module
aveLOC = the average count of lines of code (LOC) per module, and optionally
perCM = average percent of lines of comments per module (calculated by summing comments across all modules, divided by the sum of LOC of all modules) [5,12]

Oman evaluated the MI and found that the above metrics are good and sufficient predictors of maintainability [5]. Ramil suggests using *D Effort(t) = A * ModulesHandled(t) + B* where *DEffort(t)* is the effort in person-months applied during a one-month interval (from month *t* to *t+1)* and *ModulesHandled(t)* is the number of modules that were either added to the system, modified, or both (if both, the module is counted only once) during the interval [6]. The model parameters *A* and B are to be derived from historical data, by, for example, least squares regression [13]. By detecting changes to *A* and *B* as a system evolves one may infer changes in evolvabilit*y*. Polo, Piattini and Ruiz use number of modification requests, mean effort in hours per modification request, and type of correction to examine maintainability [7]. They found no meaningful influence of size metrics on the number of faults and failures. They looked at three types of maintenance requests: urgent corrective, non-urgent corrective, and perfective.

One can imagine that Ramil's model would be impacted by the semantic and syntactic size of the changes made to ModulesHandled. Similarly, a more detailed characterization of the urgent and non-urgent corrective maintenance requests in Polo, Piattini, and Ruiz's work could have led to different results.

Comparing Maintenance Techniques

When empirically comparing maintenance techniques, we sometimes intentionally change the software to study the resulting maintainability, much as testing researchers seed faults in programs to compare the effectiveness of testing techniques. When we make changes to programs under study, we often insert faults that are syntactically small. Many of the common mistakes made by Java programmers are syntactically small [9]. These changes tend to be simpler to define and manage. Thus, a more effective consideration would be semantic size.

Consider the case for testing. If we insert a large semantic fault, the fault will be easy to detect via testing (as many inputs will cause an incorrect output mapping). If we are measuring testing effectiveness, we are biasing our results toward the testing strategy. If we are comparing two testing strategies, we will be less likely to detect any difference. Conversely, if we seed faults that are too small semantically, we bias the results against the testing strategy. Or if comparing two, neither will likely work very well [1].

The same holds true for evaluation of maintenance techniques. Suppose a new maintenance technique will be used to generate test cases for changes made to the code as well as to "automatically" insert in-line comments describing the changes. If we make semantically large changes to the programs in our empirical evaluation, the technique will probably perform very well. If we make changes that are semantically very small, the technique will not perform as well. Thus, the semantic size of the changes can bias the results.

Similarly, if we are comparing technique A to technique B, the use of very small semantic changes will result in neither technique working well, while large semantic changes will result in very little differences between the two. Obviously if we make semantically small changes in the control and semantically large changes in the experimental group, our experiment is no longer comparing apples to apples. In other words, an additional threat to validity must be considered when designing experiments on corrective maintenance – have we treated the control and experimental groups equally with respect to semantic fault size?

Change Impact Analysis

Goradia [10] has suggested a technique called impact analysis that estimates, for a given test case and statement, the "impact" that statement has on the output of the program. We suggest that this is related to the semantic model in the following sense. If a statement has a large impact on the program's output when averaged over a number of test cases, then faults that appear on or partially on that statement will tend to have a larger semantic size.

When performing change impact analysis, we may want to examine the "impact" before we make a corrective maintenance change. If a small semantic change is being made, we may want to add special checks to our inspections and reviews as we know testing is not likely to uncover any small semantic mistakes we might make when making a small semantic change. Experimentation is needed to determine if impact can be used to approximate size.

2.3 Estimating Semantic and Syntactic Size of Changes

As discussed in Section 2.1, the syntactic characteristic of change deals with the number of tokens or statements altered in order to make the change. This also offers a very straightforward measurement – the number of tokens or statements that are modified approximates the syntactic size of the change. We often estimate the number of lines of code or number of classes that must be modified to correct a problem report. This characteristic is already considered often in our field. However, the <u>semantic size</u> of a change probably has a greater affect on the difficulty of the change.

This brings up several questions. Is it possible to estimate the number of inputs for which incorrect output mapping occurs from a problem report or maintenance request? It has been our experience that problem reports rarely provide this level of detail. In the rare case that we are fortunate enough to stumble across a problem report that tells us "`change value1 >= constant1 to value1 > constant1`" we know that the semantic size of the change is 1 (for the input that is equal to `constant1`). A quick look at Bugzilla [11], the bug tracking system of the open source web browser Mozilla, indicates this intuition is correct. None of the bug report descriptions we examined contains such levels of detail.

Can testing provide some assistance in this estimation? We hypothesize that there may be three ways to approximately measure the semantic size of seeded faults, which could also be applied to naturally occurring faults (i.e., maintenance changes):

- Look at the test cases used in a study and see how many tests cases find each fault
- Generate many random test cases and count how many test cases find each fault
- Obtain inputs following a usage profile and count how many test cases find each fault [1]

It appears that only the first option above can be easily adapted to corrective maintenance. We hypothesize that this can be adapted to changes (versus faults) in this way:

- Look at the test cases used and/or user scenarios used and see how many detected the fault that is now being corrected
- Count the number of reviewers or inspectors that detected the fault (that is now being corrected) during a code walkthrough or inspection

Further research and experimentation is required to determine other means for estimating the semantic size of changes or planned changes.

3 CONCLUSIONS

This paper proposes a model for characterizing changes based on the syntactic and semantic size and looks at possible implications of this model. Empirical studies in computer software maintenance should consider the characterization of changes based on its syntactic and semantic size. Techniques for estimating and measuring the semantic size of changes need to be developed. These techniques will allow a new model for evaluating maintainability, which will allow empirical investigations to help answer questions such as: Is maintainability impacted by the characteristics and size of changes? Is it easier to make a large syntactic change that is semantically small? Is it easier to make a small syntactic change that is semantically large? Other questions will follow as this area is explored in more detail.

Also, the ideas presented in this position paper need to be extended to apply to other types of maintenance such as adaptive and preventive.

4 REFERENCES

[1] Offutt, J. and Hayes, J. Huffman. A semantic model of program faults. The Proceedings of the International Symposium on Software Testing and Analysis. ACM, San Diego, California, January 1996, 195-200.

[2] De Lucia, A., Persico, A., Pompella, E., and Stefanucci, S. Improving corrective maintenance effort prediction: an empirical study. Seventh IEEE Workshop on Empirical Studies of Software Maintenance, Florence, Italy, November 9, 2001.

[3] Bianchi, A., Caivano, D., Lanubile, F., Rago, F., and Visaggio, G. Distributed and colocated projects: A comparison. Seventh IEEE Workshop on Empirical Studies of Software Maintenance, Florence, Italy, November 9, 2001.

[4] Welker, K.D. and Oman, P.W. Software Maintainability Metrics Models in Practice, *Journal of Defense Software Engineering*, 8, 11 (November/December 1995):19-23.

[5] Oman, P. and Hagemeister, J. Construction and Validation of Polynomials for Predicting Software Maintainability (92-01TR). Moscow, ID: Software Engineering Test Lab, University of Idaho, 1992.

[6] Ramil, JF. Why COCOMO Works' Revisited or Feedback Control as a Cost Factor, *Pre-Prints FEAST 2000 Workshop*, Imp. Col., London, 10 -12 Jul. 2000, pp 89 – 94.

[7] Macario Polo, Mario Piattini and Francisco Ruiz, Using code metrics to predict maintenance of legacy programs: A case study, *Proceedings of the International Conference on Software Maintenance*, 2001, 7-9 Nov. 2001, Florence, Italy.

[8] Hayes, J. Huffman. The Observe-Mine-Adopt (OMA) Model: An Agile Way to Enhance Software Maintainability. University of Kentucky Computer Science Department Technical Report 340-02, March 2002.

[9] Reilly, D. Top ten errors Java programmers make. http://www.javacoffeebreak.com/articles/toptenerrors.html.

[10] Goradia, T. Dynamic impact analysis: A cost-effective technique to enforce error-propagation. In Proceedings of the 1993 International Symposium on Software Testing, and Analysis, pages 171-181,Cambridge MA, June 1993.

[11] Bugzilla, the Bug Reporting System for Mozilla. http://bugzilla.mozilla.org/.

[12] Maintainability Index Technique for Measuring Program Maintainability, http://www.sei.cmu.edu/activities/str/descriptions/mitmpm_body.html.

[13] Gujarati, DN. *Basic Econometric*s, 3rd. Edition, Mc Graw Hill Inc., New York, 1995, 838 pp.