An Experimental Comparison of the Effectiveness of Statement Coverage and Branch Coverage Criteria

Venkata Sreenivasa Rao N., Atul Gupta, Pankaj Jalote
Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur, INDIA 208016
Email : {jalote}@cse.iitk.ac.in Fax : +91-512-590725/590413

Abstract

Over the years many studies and experiments have been done to study the effectiveness of different testing approaches. In this paper an experiment comparing the effectiveness of statement coverage and branch coverage criteria was performed. Using a coverage analyzer, five programs of moderate size are tested in a more generalized environment setup. We applied the statement coverage criteria and branch coverage criteria on each program and compared the effectiveness. The results indicate that both methods have the same fault detection ability, but branch coverage is better in terms of time taken to carry out testing as well as in time taken in locating faults compared to statement coverage.

Index Terms : Statement coverage, Branch coverage, Software testing, Experiment, Statistical analysis

told that measuring the effectiveness of testing is generally not possible, but that comparison is[17]. One classification of the test adequate criteria is by underlying testing approach where there are three basic approaches to software testing; structural testing, fault based testing and error based testing.

Two mains groups of program based structural testing are control flow testing and data flow testing. The structural testing approaches are based on the flow graph model of program structure. A basic requirement of many coverage criteria is that all nodes and edges in the program are covered by test executions. But all nodes or edges executions can not always be achieved because of possible existence of infeasible statements or edges, that is dead code. To identify whether a piece of code is dead code, is undecidable[8, 9]. Because of this unreachability we can not always attain 100% coverage of the program in terms of statements and branches.

1 Introduction

More than last two and half decades have seen rapid growth of research in software testing, design and analysis of experiments. Experimentation in Software Engineering supports the advancement of the field through an iterative learning process and software testing is no exception [2, 1]. In the mid 70’s Goodenough and Gerhart [10] put forward perhaps the most important question in software testing and that is “what is a test criteria for an adequate testing?”. Since then testing criteria has been a major research focus. A great number of criteria have been proposed and investigated[12]. E. J. Weyuker
code reading, functional testing and statement coverage method in three distinct phases involving 74 subjects (testers) of different background with four programs. The programs were having natural as well as artificial faults. In that experiment, fault detecting abilities, efforts with respect to various classes of faults and subjects were compared. In an similar replicated empirical study of effectiveness of three testing methods, code reading, functional testing and branch coverage M. Wood et. al.[19] have identified that over all effectiveness of these testing methods is similar in terms of observing failures and finding faults but their relative effectiveness depends on the nature of the program and it’s faults.

Frankl and Weiss[6] used another approach to addressing the potential invalidating factors associated with test data. They compared branch adequacy and all-uses data flow adequacy criteria using 9 small programs of different areas. Instead of using one adequate test suite for each criteria, they generated a large number of adequate test suites and used proportion of the test suites that detect errors as an estimate of the probability of detecting errors.

In another study E. J. Weyuker, et. al.[18] have identified that in comparing testing methods the probabilistic comparison has better diagnostics than analytical comparison of testing methods.

It has been pointed out that branch coverage criteria is stronger than statement coverage because if all edges in a flow graph are covered, all nodes are necessarily covered. Therefore a test suite that satisfies the branch coverage must also satisfy statement coverage. That is branch coverage subsumes statement coverage. Interestingly, Frankl and Weyuker[8] proved that the fact that C1 SUBSUMES C2 does not always guarantee that C1 is better at detecting faults.

A number of studies are performed to compare and evaluate different testing strategies, though statement and branch coverage has not been compared experimentally. As theoretically branch coverage should do better than statement coverage, one might choose this. However, statement coverage analyzers are generally easier to build and results are easier to interpret. Hence, if in practice two are similar, then one may choose to use statement coverage only. With this motivation we have identified goals of our study and performed this experiment for comparing fault detecting abilities and testing efforts required by two methods.

The rest of the paper is organized as: In section 2 the experimental setup, in section 3 data collection and analysis method used to compare are explained, results of the experiments are presented in section 4 and summary and future work are given in section 5.

2 Experimental setup

2.1 Goals

The goals of our experiment are to answer the following questions.

- Which coverage criteria has more fault detection ability, either Statement coverage or Branch coverage?

- Which coverage criteria needs more testing effort, either Statement coverage or Branch coverage?

- By limiting the testing effort viz, test suite size, testing time, which criteria is better?

- Is there any saturation effect\(^1\) for these two adequacy criteria? If there is saturation effect then at which percentage of coverage is it happening?

- Is there any new bugs which are identified through statement coverage or branch coverage methods which are not predicted? Are there any common faults identified by both methods?

2.2 Definitions of the Adequacy Criteria

To answer the questions above we need suitable metrics. To measure the fault detection ability, we use the percentage of faults that are revealed. By keeping it as percentage, it helps generalize across programs. This percent can be calculated as follows. Criteria used to measure fault detection ability i.e. percentage of faults detected is

\[
\text{fraction of faults detected} = \frac{\text{number of faults that are revealed}}{\text{total number of faults present in the program}} \times 100\%
\]

\(^1\) It is the limit at which even though we increase the testing effort no more faults are revealed
To understand the connection with effort, we use number of faults detected per unit time and percentage of distinct fault revealing test cases. Number of faults per unit time is defined as

\[
\text{Number of faults per unit time} = \frac{\text{total number of faults detected}}{\text{total time spent in testing}}
\]

Percentage of distinct fault revealing test cases

\[
\text{Percentage of distinct fault revealing test cases} = \frac{\text{Number of distinct fault revealing test cases}}{\text{Total Number of test cases used}} \times 100\%
\]

The criteria used to measure testing effort are: Number of test cases needed to satisfy the testing criteria and total time spent in testing. Total testing time is the sum of the times spent in preparing the test suite, executing the program which is to be tested and identifying defects. Time spent in preparing test suite and for locating faults separately can also be useful.

### 2.3 Testing Methodology

While testing a program using coverage criteria following steps are followed by subjects.

1. **Step 1**: Prepare the initial test case randomly by looking at the specifications of the program.
2. **Step 2**: Execute the program with the test case prepared.
3. **Step 3**: If any failure is identified then go to step 4, otherwise go to step 6.
4. **Step 4**: Use coverage information to locate the fault.
5. **Step 5**: Document the fault. If testing can not be proceeded without correcting this fault then correct the fault.
6. **Step 6**: If desired coverage is achieved then stop testing, otherwise using coverage information prepare a test case to achieve more coverage. Go to step 2.

**Test suites**: Different test suites leads to different kind of conclusions. Here we are preparing the test cases which are guided by the coverage information.

**Subjects**: This factor is a potential source of data variability. To reduce the effect of this we are considering the subjects having equal experience. Each subject is having 3 years of experience in software development.

---

**Order of runs**: The order of runs should be randomized [4]. Here we had generated random numbers and according to them experimental runs are carried out.

### 2.4 Test Programs

The experiment depends on the size (lines of code) of the software. If we take small programs[13] then the results obtained are not widely applicable. We are, therefore, taking all moderately sized unit programs ranging 200 to 350 lines of code. In practice structural testing techniques like coverage analysis are used on unit sized programs only.

The experiment depends on the programs used for testing. The programs are taken from diversified domains with general purpose uses as to predict our results to be widely applicable. Programs are written in Java, which is the most widely applicable language and we assume that because of generality of our candidate programs the results of this experiment can be well ported to the programs written in other languages.

The errors in the programs are natural committed errors by their authors and we have not introduced any artificial fault in those programs to keep general course of the experiment. If we insert the faults artificially then we might not get acceptable results. In this section we are giving brief description about test programs used.

<table>
<thead>
<tr>
<th>Sl. No.</th>
<th>Program Name</th>
<th>Noncommenting LOC</th>
<th># of Faults</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Array</td>
<td>268</td>
<td>11</td>
</tr>
<tr>
<td>2</td>
<td>Matrix</td>
<td>283</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>Sorting</td>
<td>252</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Student</td>
<td>333</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>TextFormatter</td>
<td>202</td>
<td>6</td>
</tr>
</tbody>
</table>

- **Array.java**
  This program takes list of integers as command line argument and prints the integers, search for a specified element, deletes an element specified, appends an element and inserts any integer in specified position.

- **Matrix.java**
  This program takes two matrices as input and can multiply those two matrices, add and subtract the two given matrices.
3 Data Collection and Analysis

From the data collected as in table 1, data for various adequacy criteria at different coverage percentage are computed. For illustration, table 2 is showing percentage of faults detected at different coverage.

3.1 Mathematical Model

For carrying out testing, five programs are selected in this design. Lack of homogeneity between the programs will contribute to the variability of effectiveness parameter measurements and will tend to inflate the experimental setup. Here, therefore we will use pairwise comparison design to identify the effectiveness of the two methods.

Our statistical model that describes the data from the experiment is

\[ y_{ij} = \mu_i + \beta_j + \epsilon_{ij} \]

where \( y_{ij} \) is the observation on the effectiveness parameter for method \( i \) (statement or branch coverage) on program \( j \), \( \mu_i \) is the true mean for some effectiveness parameter in observation for \( i^{th} \) method and \( \epsilon_{ij} \) is a random experimental error with mean zero and some variance \( \sigma_i^2 \).

3.2 Hypothesis to be tested

For all the effectiveness parameters in consideration null hypothesis \( H_0 : \mu_1 = \mu_2 \) is tested using pairwise comparison. Testing \( H_0 : \mu_1 = \mu_2 \) is equivalent to testing

\[
H_0 : \mu_\delta = 0 \\
H_1 : \mu_\delta \neq 0
\]

The test statistic for this hypothesis is

\[
t_0 = \frac{d}{S_d/\sqrt{n}}
\]

where

\[
d = \frac{1}{n} \sum_{j=1}^{n} d_j
\]

is the sample mean of the differences and

\[
S_d = \left[ \frac{\sum_{j=1}^{n} (d_j - \bar{d})^2}{n - 1} \right]^{1/2}
\]

is the sample standard deviation of the differences. \( H_0 : \mu_\delta = 0 \) would be rejected if \( |t_0| > t_{\alpha/2, n-1} \) for some \( \alpha \) which is the probability of type I error.

3.3 Data Collection

Experimental data for each program, for each coverage method is collected using following form.

3.4 Computation of test statistics

Test statistics for different effectiveness criteria are computed as given in section 2.2. Table 3 is showing the results of \( t \)-Test with \( \alpha = 0.05 \). We have computed confidence intervals for the hypothesis which was rejected at some specified value of \( \alpha \). It is shown in the last column of the table 3.

3.5 Graphical analysis

The graphs are drawn which show the performance of the two methods taken as mean of computed value of effectiveness criteria under consideration with respect to the percentage of coverage achieved. They are shown in figures 1, 2, 3, 4.

4
Name of the Subject:
Experience of the Subject:
Program Name:
Coverage Analysis Method using (Statement/Branch):

<table>
<thead>
<tr>
<th>Test No.</th>
<th>time spent in designing test case</th>
<th>Coverage percentage achieved</th>
<th>Line no</th>
<th>Brief description of fault</th>
<th>time spent in locating fault</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Is the coverage information is helpful in locating the bug?

Is the coverage feedback enough?

Any suggestions regarding the coverage feedback?

After achieving 100% coverage do you think that all faults are identified?

How much percentage coverage is enough?

Table 1: Data Collection Form

![Graph 1: Coverage Percentage Vs Percentage of distinct fault revealing test cases](image1)

![Graph 2: Coverage Percentage Vs Percentage of faults revealed](image2)

Figure 1: Coverage Percentage Vs Percentage of distinct fault revealing test cases

Figure 2: Coverage Percentage Vs Percentage of faults revealed

4 Results

The criteria used for fault detecting ability is percentage of faults that are revealed. We obtained that both methods identify same number of faults. The result of the hypothesis testing for this criteria also shows that there are not enough evidences to reject the null hypothesis, the equality of the means of percentage of faults that are revealed.

The criteria which related fault detection abilities with efforts are: 1. number of faults identified per unit time, 2. percentage of fault revealing test cases and
### Table 3: Pairwise comparison of effectiveness criteria

<table>
<thead>
<tr>
<th>Sl. No</th>
<th>Effectiveness Parameter</th>
<th>Null Hypothesis</th>
<th>t-value</th>
<th>t-crit</th>
<th>P-value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>% of faults identified</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>0.091628</td>
<td>2.063898</td>
<td>0.927754</td>
<td>$\mu_1 = \mu_2$</td>
</tr>
<tr>
<td>2.</td>
<td># of Faults detected per unit time</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>-1.74</td>
<td>2.063898</td>
<td>0.092</td>
<td>$\mu_1 = \mu_2$</td>
</tr>
<tr>
<td>3.</td>
<td>% of distinct fault revealing test cases</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>-2.608284</td>
<td>2.063898</td>
<td>0.015413</td>
<td>$\mu_1 &lt; \mu_2$</td>
</tr>
<tr>
<td>4.</td>
<td>Number of test cases</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>2.68544</td>
<td>2.063898</td>
<td>0.01293</td>
<td>$\mu_1 &gt; \mu_2$</td>
</tr>
<tr>
<td>5.</td>
<td>Total time taken (min)</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>2.868969</td>
<td>2.063898</td>
<td>0.008454</td>
<td>$\mu_1 &gt; \mu_2$</td>
</tr>
<tr>
<td>6.</td>
<td>Total time spent in preparing test suite</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>2.168995</td>
<td>2.063898</td>
<td>0.040214</td>
<td>$\mu_1 &gt; \mu_2$</td>
</tr>
<tr>
<td>7.</td>
<td>Total time spent in locating faults (min)</td>
<td>$\mu_1 - \mu_2 = 0$</td>
<td>2.611727</td>
<td>2.063898</td>
<td>0.015293</td>
<td>$\mu_1 &gt; \mu_2$</td>
</tr>
</tbody>
</table>

Figure 3: Coverage Percentage Vs Faults detected per unit time (100 minutes)

Figure 4: Coverage Percentage Vs Total time (min)

3. number of test cases used in testing. The analysis of paired t-Test has identified that in case of number of faults detected per unit time, the ability of the two methods are similar. But, there are some evidences that the branch coverage is having an edge over statement coverage (see the corresponding p-value in table 3). Also there are enough evidences that the effectiveness of branch coverage for percentage of fault revealing test cases is clearly shown to be better than statement coverage. Another criteria in this regard is number of test cases used for the testing, and our hypothesis that the two criteria means is same is rejected and we conclude that here also branch coverage is taking less number of test cases for testing as compared to statement coverage.

The efforts required by the testing can be measured directly by measuring the time taken by testing methods in designing, executing test cases and time required to locate the faults. Our results for total testing time, time required for designing test cases and time required to locate the fault are evidently less
<table>
<thead>
<tr>
<th>percent coverage achieved</th>
<th>Program</th>
<th>percentage of faults</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Statement Coverage</td>
</tr>
<tr>
<td>51-60%</td>
<td>Array</td>
<td>81.81</td>
</tr>
<tr>
<td></td>
<td>Matrix</td>
<td>46.15</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>37.50</td>
</tr>
<tr>
<td></td>
<td>Student</td>
<td>72.72</td>
</tr>
<tr>
<td></td>
<td>TextFormatter</td>
<td>50</td>
</tr>
<tr>
<td>61-70%</td>
<td>Array</td>
<td>81.81</td>
</tr>
<tr>
<td></td>
<td>Matrix</td>
<td>61.53</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>62.50</td>
</tr>
<tr>
<td></td>
<td>Student</td>
<td>81.81</td>
</tr>
<tr>
<td></td>
<td>TextFormatter</td>
<td>50</td>
</tr>
<tr>
<td>71-80%</td>
<td>Array</td>
<td>90.90</td>
</tr>
<tr>
<td></td>
<td>Matrix</td>
<td>69.23</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>62.50</td>
</tr>
<tr>
<td></td>
<td>Student</td>
<td>90.90</td>
</tr>
<tr>
<td></td>
<td>TextFormatter</td>
<td>66.66</td>
</tr>
<tr>
<td>81-90%</td>
<td>Array</td>
<td>90.90</td>
</tr>
<tr>
<td></td>
<td>Matrix</td>
<td>84.61</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>62.50</td>
</tr>
<tr>
<td></td>
<td>Student</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>TextFormatter</td>
<td>83.33</td>
</tr>
<tr>
<td>91-100%</td>
<td>Array</td>
<td>90.90</td>
</tr>
<tr>
<td></td>
<td>Matrix</td>
<td>92.30</td>
</tr>
<tr>
<td></td>
<td>Sorting</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>Student</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td>TextFormatter</td>
<td>83.33</td>
</tr>
</tbody>
</table>

Table 2: Percentage of faults detected.

We have, therefore, carried out experimental comparison of statement coverage criteria and branch coverage criteria in a generalized environment. In this experiment we have tested five moderately sized Java programs using statement coverage and branch coverage testing approaches and compared their effectiveness in terms of suitable criteria for fault detection abilities and efforts requirements, using a paired comparison design methodology. Our study shows that fault detection ability of the two criteria are similar, but when compared with efforts required in terms of number of test cases used, percentage of fault revealing test cases and time required for testing, branch coverage indicates better performance than statement coverage.

One limitation of our study is that the results are applicable to subjects (i.e. testers) having five years of software development experience.

This work can be extended in comparing the reliability of the softwares, MTTF, MTBF which are tested using the two criteria, statement coverage and branch coverage. The same experiment can be carried out with large sized softwares to make results more general.

### References


