Does code coverage provide a good stopping rule for operational profile based testing?

Breno Miranda[*†]
[*]Università di Pisa
Largo B. Pontecorvo, 3 - 56127
Pisa, Italy
{firstname.lastname}@di.unipi.it

Antonia Bertolino[†]
[†]ISTI - CNR
Via Moruzzi 1 - 56124
Pisa, Italy
{firstname.lastname}@isti.cnr.it

ABSTRACT

We introduce a new coverage measure, called the operational coverage, which is customized to the usage profile (count spectrum) of the entities to be covered. Operational coverage is proposed as an adequacy criterion for operational profile based testing, i.e., to assess the thoroughness of a black box test suite derived from the operational profile. To validate the approach we study the correlation between operational coverage of branches, statements, and functions, and the probability that the next test input will not fail. On the three subjects considered, we observed a moderate correlation in all cases (except a low correlation for function coverage for one subject), and consistently better results than traditional coverage measure.

CCS Concepts

•**Software and its engineering** → **Software testing and debugging;** •**General and reference** → *Verification;* Metrics;

Keywords

Coverage Testing, Operational Coverage, Operational Profile Based Testing, Program Spectra, Relative Coverage

1. INTRODUCTION

While all researchers agree that test coverage measures can be useful in pinpointing portions of code that have not been tested, the relation between test coverage and test effectiveness is highly controversial, and has been the subject of countless analytical and empirical studies (see, e.g., [4, 24], among the earliest ones). After almost three decades, the debate on the topic does not seem to decline, and current testing research still seeks an answer to questions such as "Is Branch Coverage a Good Measure of Testing Effectiveness?" [23]. A recent experiment on large scale [11] concludes that high coverage measures achieved by a test suite do not necessarily indicate that the latter also yields high effectiveness. Thus, generating test cases for coverage as a target may be risky, as warned from many sides (e.g., [15, 22]).

However, coverage measure used as a supplement to other non-coverage-based testing methods can be an effective tool [22], for example to decide whether a test suite derived using another black-box method is adequate.

This paper goes in this direction and investigates *the use of code coverage measures as a stopping rule for operational profile based testing.*

Operational profile based testing is grounded on the notion that not all faults have the same importance. Depending on how it will be exercised by the users, a program can show quite different levels of reliability [14].

On the other side, almost all studies assessing the effectiveness of coverage testing have used as a measure of effectiveness the faults (or mutations) detected without distinguishing their probability of failure in use (e.g., [11, 13, 22, 23, 24], just to cite a few). Thus, we still know little about how coverage testing is related or not to *delivered reliability* [8]. An exception is the study by Del Frate and coauthors [5], who observed that the relation between coverage and reliability varied widely with subject's size and complexity.

All studies made so far (concerned with either faults or mutants detected or with delivered reliability) considered traditional coverage measures, which require that *all* entities are covered *at least once*. In other terms, all entities are considered as having same relevance for the purpose of completing coverage. This assumption seems contradictory with the very idea of reliability according to which user's functions should be exercised more or less frequently accordingly to user profiles, and in doing this, code entities as well will consequently be exercised with different frequencies.

In our research, we have been investigating for some time novel coverage measures that customize coverage to user's relevance [16, 18]. In particular, in previous work we distinguished between relevant entities and not relevant ones depending on the usage domain.

In this paper, for the first time, we not only distinguish between relevant and not relevant entities, but also, among relevant entities, we take into account how much they are exercised, i.e., we also distinguish entities that are very often exercised from those that are scarcely exercised. We do this to reflect the profile of usage into the measure of coverage. Our intuition is that this customized coverage could better measure the adequacy of an operation profile based test suite.

AST '16 Austin, Texas USA

DOI: http://dx.doi.org/10.1145/2896921.2896934

Profiling of code entities according to different usage profiles is referred to as program spectra [10]. The idea of using program spectra in software engineering tasks is not new: program spectra have been used, among others, for regression testing [25] and fault localization [1]. To the best of our knowledge, however, our research is the first attempt to tune coverage measures based on program spectra, for the purpose of reflecting the importance of program entities.

The preliminary results we obtained seem to sustain our intuition that spectra-based coverage may be taken as a stopping rule for operation profile based testing, better than traditional coverage. Such conclusion can be most usefully applied when the developer can leverage field data collected from profiling usage of the instrumented software (like the scenario described in [20]). Our proposed coverage measure may improve scalability of automated testing based on field data, because it provides an evaluation of test thoroughness that is customized to the usage profile, and thus testing resources can be better calibrated.

In summary, the contributions of the paper include:

- a novel method to measuring code coverage that exploits program spectra: this is the first time that code entities are profiled in order to customize test coverage
- the design of the first of its kind study of using (operational profile based) coverage as an adequacy criterion for operational profile based testing (black box)

The rest of the paper is structured as follows: in the next section we introduce operational coverage. Then in Section 3 we present the design of the validation study and in Section 4 we illustrate and discuss the results. Related work and Conclusions sections complete the paper.

2. OPERATIONAL PROFILE BASED COVERAGE

Our proposed approach is meant to provide a practical stopping rule for operational profile based testing. An operational profile provides a quantitative characterization of how a system is used [19]. Software testing based on the operational profile ensures that testing resources are focused on the most frequently used operations, and thus maximizes the reliability level that is achievable within the available testing time [19]. So, it can be a good testing strategy when safety is not an issue.

As we start from the assumption that the developer adopted an operational profile based testing strategy, we also can assume that either an operational profile is derived by domain experts during the specification stage, or that, as schematized in Figure 1, this profile is obtained from real world usage, e.g., by monitoring field data by means of an infrastructure such as Gamma [21].

If this developer selects a test suite from the operational profile, how can they decide whether the test suite is adequate and testing can be stopped, or otherwise more test cases should be derived? (see Figure 1) This is where our approach can help. It foresees two main steps:

1. Classify the entities according to their importance to the operational profile under testing
2. Calculate operational coverage based on the importance of the entities covered

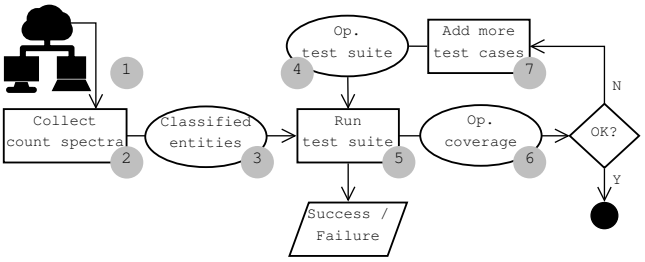

Figure 1: Overview of the approach

We rate the importance of entities based on their frequency of usage, i.e., we make use of program spectra [10]. A program spectrum characterizes a program's behavior by recording the set of entities that are exercised as the program executes. In this work we investigated coverage of three types of entities and correspondingly adopted three types of spectra:

- **Branch-count spectrum (BCS):** for each conditional branch in a given program P, the spectrum indicates the number of times that branch was executed.
- **Statement-count spectrum (SCS):** for each statement in a given program P, the spectrum indicates the number of times that statement was executed.
- **Function-count spectrum (FCS):** for each function in a given program P, the spectrum indicates the number of times that function was executed.

Based on the spectra, we then classify the entities (step 1 of our approach) into different importance groups. In this work we used three groups: *high*, *medium*, and *low*, but other different groupings could be decided. To cluster entities into group, again, different methods could be applied. In this first investigation we opted for ordering the list of entities according to their frequency and assigning the first 1/3 entities to the high frequency group; the second 1/3 entities to the medium frequency group; and the last 1/3 entities to the low frequency group. Surely, the importance of a given entity could be assigned in many different ways and the effect of choosing one approach or another should be investigated in future work.

We calculate operational coverage (step 2) by computing the weighted arithmetic mean of the rate of covered entities according to Equation 1 below. Again, we observe that there exist many other different ways in which we could calculate operational coverage and Equation 1 is just one of the many possibilities.

$$OC = \sum_{i=1}^{n} x_i w_i \quad (1)$$

where:

n = number of importance groups (3 in this paper)
x_i = the rate of covered entities from group i
w_i = the weight assigned to group i

In this work we assigned the weights for the importance groups (the w_i of Equation 1) in such a way that the *medium* group is three times more important than the *low*, and the *high* group, on its turn, is three times more important than the *medium* one.

3. EXPLORATORY STUDY

In this section we discuss the settings of our exploratory study. We focus on the following research question:

RQ: *Does operational coverage provide a good adequacy criterion (stopping rule) for operational profile based testing?*

3.1 Study Subjects

In order to carry out our exploratory study and to investigate our research questions in a realistic setting, we looked for subjects in the Software-artifact Infrastructure Repository (SIR) [6]. SIR contains a set of real, non-trivial programs that have been extensively used in previous research. For selecting our subjects, the only prerequisite was that they should contain faults (either real or seeded ones) and a test suite associated with them.

We considered three subjects — grep, gzip, and sed — as these are frequently used in academia for software testing research. grep is a command-line utility that searches for lines matching a given regular expression in the provided file(s); gzip is a software application used for file compression and decompression; and sed is a stream editor that performs basic text transformations on an input stream. grep and gzip are available from SIR with 6 sequential versions (1 baseline version and 5 variant versions with seeded faults) whereas sed contains 8 sequential versions (1 baseline version and 7 variant versions with seeded faults). Because each version contains a different number of seeded faults, for this first exploratory study we selected, from each subject, the version that contained the highest number of faults that could be revealed by our test pool (detailed next). We then proceeded with version 3 of grep, version 4 of gzip, and version 2 of sed. Because we adopted only one version of each subject, throughout the rest of this paper we refer to them by the subjects' names only without reporting the version.

Table 1 provides some additional details about the study subjects. Column "LoC" shows the lines of code[1] of each subject. The meaning of the last two columns is explained later on.

Table 1: Details about the study subjects considered in our investigations

Sub.	Ver.	LoC	# Seeded faults	# Failing test cases	# Faults revealed
grep	v3	10124	18	1664	5
gzip	v4	5233	12	1638	5
sed	v2	9867	5	3195	4
Total:		25224	35	6497	14

3.2 Operational Profile

Operational profiles can be defined in many different ways. Here, following [19], we express it as the list of operations that are expected to be invoked by users along with their associated occurrence probabilities. Ideally it is developed during system specification with the participation of the system experts (e.g.: system engineers, designers, etc) and domain experts (e.g.: analysts, customers, etc). Because such an ideal operational profile was not available for the subject systems we investigated, we ourselves defined the operational profile for each subject. We accomplished this task by getting acquainted with the system (after carefully reading the user manual for the version of the subject being investigated) and by following Musa's stepwise approach [19]. For experimental purposes, we stopped the process of creating the operational profile just before assigning the occurrence probabilities for each operation identified, as these are taken as an independent variable of our study.

Table 2 displays an excerpt of the list of operations we identified for grep. A graphical version containing the full list of operations identified (grouped according to different usage paths) is available at http://bit.ly/op_grep.

Table 2: List of operations identified for grep

ID	Description
Op_{001}	Look for a single pattern in a single input file
Op_{002}	Look for multiple patterns, obtained from a file, in a single input file
...	...
Op_{191}	Interpret the pattern as a list of fixed strings and match any of them
Op_{192}	Interpret the pattern as an extended regular expression and print the number of matching lines

3.3 Study Settings

Besides the study subjects obtained from SIR and the subjects' operational profiles developed by ourselves, for each of the three investigated subjects we also created a few additional artifacts required for our study. Some of them were derived once and for all:

Test Pool. For each subject investigated we created a test pool containing 10k test cases uniformly distributed among the operations in the subject's operational profile.

Fault Matrix. For each subject considered we run the set of 10k test cases over the baseline version first (the one without any faults enabled), and then over the faulty versions of that subject. To get a precise mapping of which test cases would reveal which faults we compile one faulty version for each seeded fault available. For grep, for example, we have one baseline version and 18 faulty versions, which accounts for 190k test cases run to generate the fault matrix. The second last column and the last column of Table 1 refer to the results of running the set of 10k test cases over the studied subjects: they display the number of failing test cases and of seeded faults that could be revealed, respectively.

Operation Matrix. Each test case in the test pool is created for a specific operation in the operational profile and this mapping (test case, operation) is stored in the operation matrix.

Other artifacts, on the other hand, were created on a "per observation" basis. For each subject, we made 500 observations, each one related to a different operational profile. More precisely, for each observation we derived the following artifacts:

Customized Operational Profiles. For deriving a customized operational profile we randomly select an arbitrary number of operations and randomly assign their respective occurrence probabilities in a way that the sum of the individual probabilities is equal to 1.

[1]Collected using CLOC (http://cloc.sourceforge.net/).

Count Spectra. For each customized operational profile we derived three different spectra: the branch-count, the statement-count, and the function-count spectrum. In order to do so, we exercise the subject with randomly generated test cases that are derived according to the operations and their respective occurrence probabilities defined in the customized operational profile. Observe that this set of randomly generated test cases is completely separated from the test pool previously introduced.

We also used the gcov[2] and lcov[3] utilities for collecting accurate coverage metrics, and our own code to automate the majority of the steps followed during this study.

3.4 Tasks and Procedures

For each subject (and for each one of the 500 customized profiles created per subject) the following tasks are performed:

1. We carry out operational profile based testing by selecting the next test case to be run (from the 10k set) according to the occurrence probabilities defined in the customized operational profile.
2. After each test case is run, we calculate:
   (a) the traditional coverage achieved
   (b) the operational coverage achieved (calculated according to Equation 1)
   (c) the probability of failure for the next test case, θ

The coverage metrics (items 2a and 2b) are calculated for the three adequacy criteria considered in this study and we stop testing if none of them increase after a sequence of 10 test cases.

We illustrate the way we calculate the probability of failure (item 2c) through a simple example pointing to the artifacts described in Section 3.3:

> *Customized operational profile:* we assume a very simple profile including only four operations: Op_1 with occurrence probability $Pr = 0.6$, Op_2 with $Pr = 0.3$, Op_3 with $Pr = 0.1$, and Op_4 with $Pr = 0.0$.
>
> *Test pool:* by construction, the test pool contains the same amount of test cases for each operation in the operational profile. For this example, we assume the test pool contains 2500 test cases per operation
>
> *Fault matrix:* we assume only one fault, Fault 1, and that from the Fault matrix we can see it is revealed by 100 test cases in the 10k pool
>
> *Operation matrix:* from the operation matrix we can match operations to fault-revealing test cases (in the fault matrix). We assume we get: 50 failing test cases for Op_1, 30 for Op_2, 0 for Op_3, and 20 for Op_4.

Then, when no test case has been run, the probability that the next test will fail is:

$$\theta_{F_1} = \frac{(50 \times 0.6) + (30 \times 0.3) + (0 \times 0.1) + (20 \times 0)}{2500} = 0.0156$$

Of course, when more faults exist, θ is the overall sum of the individual probability of failure for each fault.

[2]https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[3]http://ltp.sourceforge.net/coverage/lcov.php

4. STUDY RESULTS

A summary of the output of the tasks described in Section 3.4 is displayed in Table 3. The second column shows the span variations in the number of test cases required for performing operational profile based testing for each subject while the third column presents the average number of test cases among the 500 customized operational profiles created for our study. As we can see, for all the subjects, testing stopped after around 60 test cases. The biggest span variation happened for gzip with some operational profiles requiring as few as 13 test cases to complete testing, whereas other profiles required 136 tests; sed had the smallest variation with the number of test cases ranging from 15 to 98.

Table 3: Average traditional and operational coverage achieved per subject

Sub.	#TCs span	Avg. #TCs	Branch		Statement		Function	
			trad.	*oper.*	*trad.*	*oper.*	*trad.*	*oper.*
grep	24 to 116	64	24.7	86.3	42.4	89.2	61.5	92.5
gzip	13 to 136	56	39.9	79.3	52.0	85.4	60.3	95.5
sed	15 to 98	51	29.5	95.5	48.3	96.2	71.3	98.1
Average:		57	31.3	87.0	47.6	90.3	64.4	95.3

Table 3 also displays the average traditional and operational coverage achieved grouped by different adequacy criteria. In this table, "*trad.*" and "*oper.*" stand for traditional coverage and operational coverage, respectively. Operational coverage achieved higher coverage values in all the cases. This was expected because, by construction, operational coverage targets only a subset of the entities for each operational profile, which increases the chances of providing high coverage values.

Figure 2 shows, for all the subjects and coverage criteria investigated, the average traditional and operational coverage achieved as the number of test cases increases. The x-axes display the number of test cases while the y-axes represent the coverage achieved. In this figure, traditional coverage and operational coverage are represented by the continuous line and the dashed line, respectively. For each test case n, its equivalent point in the curve represents the average coverage (of the 500 customized profiles) achieved after n test cases. Notice that not all the profiles finished after the same amount of test cases. For this reason it is possible to see, for gzip in particular, some fluctuation in the curve when the number of test cases gets close to the maximum number of tests required for that product.

From the graphs, the main observation is the fact that the curve of operational coverage rises sharply and achieves high coverage values even after just a few test cases. This happens due to the combination of two facts: (i) the most frequently exercised entities contribute more for the computation of the coverage achieved (because of the weight assigned to the high frequency group); and (ii) the operational profile based testing strategy selects test cases that cover the most frequent entities first.

4.1 Answer to the Research Question

To answer our research question we computed, for both traditional and operational coverage, the correlation between the coverage achieved and the probability that the next test case will *not* fail ($1 - \theta$, which is the reliability of the next

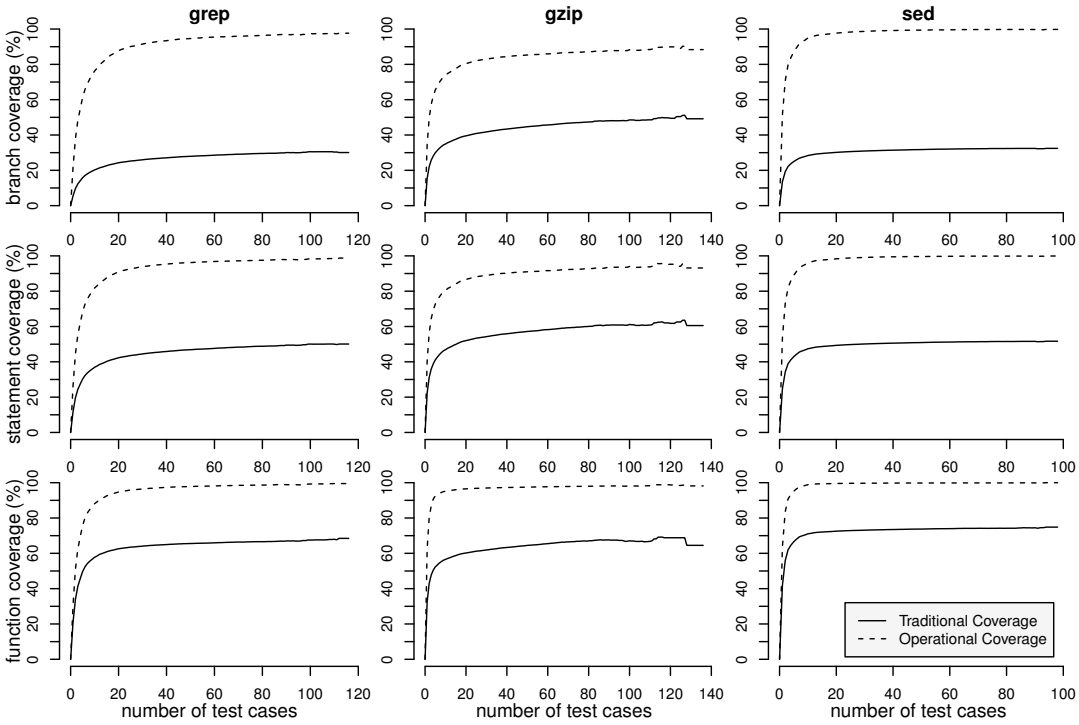

Figure 2: Average traditional and operational coverage achieved as the number of test cases increases

invocation). For doing so, we adopted the Kendall τ correlation coefficient. Kendall τ is similar to the more commonly used Pearson coefficient but it does not require the variables to be linearly related or normally distributed. By using Kendall τ we avoided introducing unnecessary assumptions about the distribution of the data.

As Kendall τ measures the similarity of the orderings of the data when ranked by each of the variables, a high correlation means that one can predict the rank of the importance of the faults revealed given the rank of the coverage achieved, which in practice is nearly as useful as predicting the absolute importance of the faults revealed.

Table 4 displays the Kendall τ correlation between the coverage achieved and the probability that the next test case will not fail, grouped by the different coverage criteria investigated. For interpreting the data in accordance with [11], we use the Guildford scale, in which correlations with absolute value less than 0.4 are described as "low", 0.4 to 0.7 as "moderate", 0.7 to 0.9 as "high", and over 0.9 as "very high".

Table 4: Kendall τ correlation between coverage and $1-\theta$ (all entries are significant at the 99.9% level)

Subject	Branch		Statement		Function	
	trad.	*oper.*	*trad.*	*oper.*	*trad.*	*oper.*
grep	0.37	**0.40**	0.38	**0.41**	0.39	0.35
gzip	0.41	**0.45**	0.44	**0.46**	0.39	**0.44**
sed	0.39	**0.50**	0.40	**0.52**	0.35	**0.47**
Average:	0.39	**0.45**	0.41	**0.46**	0.38	**0.42**

As we can see, operational coverage yielded better correlation coefficients than traditional coverage for the vast majority of the cases (we highlight in bold the cases in which operational coverage performed better than traditional coverage). The only exception was for grep when considering the function adequacy criterion, in which traditional coverage achieved a correlation coefficient of 0.39 and operational coverage produced 0.35 (statistically, they were tied as both achieved low correlation). This was the only case in which operational coverage yielded low correlation. For the remaining cases, operational coverage always achieved moderate correlation and, in 5 out of the 9 cases, it was statistically significant better than traditional coverage (not tied in the same correlation group).

Traditional coverage achieved moderate correlation 3 times and in all the cases it was statistically tied with operational coverage, if we consider the correlation group; if we consider the absolute correlation coefficient achieved, though, it was always defeated by operational coverage.

4.2 Discussion and Threats to Validity

The exploratory study showed that in the majority of the cases operational coverage is statistically better than traditional coverage as an adequacy criterion for operational profile based testing. In the following we further discuss potential costs and benefits of the approach.

On the costs of the approach. The cost of classifying the entities according to their importance (the first step of our approach) will depend on how the operational profile is derived. For the case advised in Figure 1 in which the operational profile can be derived by monitoring field data, the count spectrum (bullet 3 in Figure 1) required for defining the relevance of the entities might even be readily available, or otherwise can be obtained by using mining techniques to capture the frequencies of the entities that are being exercised by the users.

Regarding the cost of computing the operational coverage (bullet 6 in Figure 1), as for any coverage criterion, operational coverage presupposes that the code is instrumented so to allow the identification of the entities exercised. So the cost of applying the operational coverage equation is comparable to any other coverage metric.

Operational coverage vs. traditional coverage. Operational coverage may be particularly helpful for test suite augmentation. After the developer has derived the test suite to verify a given program, two main things could happen: (i) for all the code exercised by the program's users there exist at least one test case in the test suite that covers that code; or (ii) there exist some code exercised by the program's users for which there does not exist any test case in the test suite.

Operational coverage would acknowledge the former case by yielding 100% coverage. For the latter case, it would provide an assessment of the extent to which the derived test suite is adequate to that specific operational profile. Moreover, it would also provide the precise portions of code that are *actually exercised* by the program's users and that are not covered by the test suite, guiding the test suite enhancement process towards the real usage of the program.

Regarding traditional coverage, for the case (i), if the test suite does not achieve 100% traditional coverage — probably the case — the developer cannot tell whether or not all the code that is relevant to the program's users has been covered. Similarly, for case (ii) traditional coverage does not help the developer unless they are willing to augment the test suite until 100% traditional coverage is reached, which may be impractical even for small programs.

Operational coverage as selection criterion. Even though we believe that operational coverage may be a promising approach for operational profile based test case selection as well, its adoption may not be as straightforward as it was for adequacy criterion. One of the issues is that of collateral coverage: by mapping white-box entities to existing test cases there could be the risk of selecting test cases that cover the target entity but that are not related to the target operation. This could result in selecting not relevant test cases in the derived test suite. To overcome this effect, some fairly sophisticated selection approach that takes into account not only the relevance of the entities to be covered, but also the way the candidate test cases cover those entities, needs to be devised. We plan to investigate this as part of our future work.

The exploratory study results should be interpreted by keeping in mind the following potential threats to validity:

Subject representativeness: In this work we investigated the proposed operational coverage on three C programs from the SIR repository. Additional studies using a range of diversified subjects should be conducted for better representativeness.

Subjects' operational profile: Because the subjects' operational profiles were not readily available, we had to develop them ourselves. It is possible, then, that the set of operations we identified is not a complete list of operations that could be performed by real users. We controlled this threat by carefully reading the subjects' documentation to understand well how they could be used before developing the operational profiles.

Customized profiles representativeness: We derived the customized operational profiles by randomly selecting the target operations and their respective occurrence probabilities, which may have results into unrealistic profiles. To control this threat, we created 500 customized operational profiles for each subject investigated expecting that the effect of possible unrealistic profiles would be minimized with a big number of observations. Moreover, since both metrics (traditional and operational coverage) are captured for each customized profile, we do not see how such threat could produce different impacts on our evaluations in a systematic way, and thus influence the results biased to the benefit of one approach or another.

Faults representativeness: The subjects considered in our study contained seeded faults and subjects with real faults might yield different results. Control for this threat can be achieved only by conducting additional studies using subjects with real faults.

Operational coverage calculation: The operational coverage is influenced by parameters that allow a high level of customization: the importance assigned to the entities, the weight assigned to the importance groups, and the equation itself. Control for this threat can be achieved only by conducting additional studies using different configurations.

5. RELATED WORK

Code coverage criteria occupy a large part of software testing literature [9, 26]. Many authors aim at finding novel criteria that subsume existing ones or that improve fault-finding effectiveness. Here, we do not propose yet another coverage criterion by identifying a new type of control-flow or data-flow entity to be covered. We propose instead that the entities to be covered (even basic ones, such as function, statement or branch) be weighted based on their relevance according to a usage profile.

Our research is inspired by the idea of relative coverage originally defined in [3]. In our previous work [16, 18, 17] we distinguished between relevant and not relevant coverage, which amounted to give entities either a 1 or 0 weight, i.e., we used a hit spectrum. In this paper for the first time we considered frequency of coverage and accordingly propose to weight entities using a count spectrum.

As discussed in the introduction, the effectiveness of coverage criteria is still a very active research topic [22, 11, 13]. However, the studies evaluating the effectiveness of coverage measures consider the faults (or mutations) has having same importance, and do not take into account their respective probability of failure. Considering the possible impact of faults detected, as we do here, provides a more meaningful picture when the software under test is going to be used under different profiles.

Program spectra[10] have been used extensively in software analysis. Beyond the original application in program optimization [2], more recently code profiling information has been used to analyze the executions of different versions of code, e.g. in regression testing [25], and to compare traces of failed and successful runs in fault diagnosis [1]. Here we propose to exploit traces information to tune coverage measures onto the usage profile. This is a novel application of spectra in operational profile based testing that has never been tried, and could be exploited in many ways.

Modern pervasive and interconnected networks and huge advances in potential to collect and analyze big data make it

thinkable that developers continue testing and maintenance of deployed software by exploiting usage profiling information. This requires the establishment of proper infrastructures [21], but can provide many opportunities for improved testing and analysis techniques [7]. For example, in [20] field data are exploited for impact analysis and regression testing; in [12] field failures data are used to support in-house debugging and fault localization. Our approach is one among the many opportunities provided by field data to improve testing techniques.

6. CONCLUSIONS AND FUTURE WORK

We have introduced operational coverage, which measures code coverage taking into account whether and how the entities are relevant with respect to a user's operational profile. Our study showed that operational coverage is better correlated than traditional coverage with the probability that the next test case derived according to the user's profile will not fail.

Operational coverage is the very first attempt to tune coverage testing based on program count spectra. In particular, here we used a simple approach to map count spectrum of branches, statements, and functions into a coverage measure. However, we believe that the very idea introduced here paves the way to exploring many other powerful measures. In Section 4.2 we have already pointed out several directions for future research, in particular more empirical studies are of course required, and potential usage of spectra-based coverage for test selection seems an interesting challenge.

7. ACKNOWLEDGMENTS

Breno Miranda wishes to thank the Brazilian National Council for Scientific and Technological Development (CNPq) for providing his scholarship grant.

8. REFERENCES

[1] R. Abreu, P. Zoeteweij, R. Golsteijn, and A. J. Van Gemund. A practical evaluation of spectrum-based fault localization. *Journal of Systems and Software*, 82(11):1780–1792, 2009.

[2] T. Ball, P. Mataga, and M. Sagiv. Edge profiling versus path profiling: The showdown. In *Proc. 25th Symp. on Principles of Prog. Languages*, POPL'98, pages 134–148, New York, NY, USA, 1998. ACM.

[3] C. Bartolini, A. Bertolino, S. Elbaum, and E. Marchetti. Whitening SOA testing. In *Proc. ESEC/FSE '09*, pages 161–170. ACM, 2009.

[4] V. Basili and R. Selby. Comparing the effectiveness of software testing strategies. *IEEE Trans. Softw. Eng.*, SE-13(12):1278–1296, Dec 1987.

[5] F. Del Frate, P. Garg, A. Mathur, and A. Pasquini. On the correlation between code coverage and software reliability. In *Proc. 6th Int. Symposium on Sw Reliability Engineering*, pages 124–132, Oct 1995.

[6] H. Do, S. G. Elbaum, and G. Rothermel. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empirical Software Engineering*, 10(4):405–435, 2005.

[7] S. Elbaum and M. Diep. Profiling deployed software: Assessing strategies and testing opportunities. *IEEE Trans. Softw. Eng.*, 31(4):312–327, Apr. 2005.

[8] P. Frankl, R. Hamlet, B. Littlewood, and L. Strigini. Evaluating testing methods by delivered reliability. *IEEE Trans. Softw. Eng.*, 24(8):586–601, Aug 1998.

[9] P. G. Frankl and E. J. Weyuker. An applicable family of data flow testing criteria. *IEEE Trans. Softw. Eng.*, 14(10):1483–1498, 1988.

[10] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. In *Proc. PASTE '98*, pages 83–90, 1998.

[11] L. Inozemtseva and R. Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*, pages 435–445. ACM, 2014.

[12] W. Jin and A. Orso. F3: fault localization for field failures. In *Proc. ISSTA*, pages 213–223. ACM, 2013.

[13] P. Kochhar, F. Thung, and D. Lo. Code coverage and test suite effectiveness: Empirical study with real bugs in large systems. In *Proc. SANER*, pages 560–564, March 2015.

[14] M. R. Lyu et al. *Handbook of software reliability engineering*. IEEE computer society press CA, 1996.

[15] B. Marick. How to misuse code coverage. In *Proc. 16th Int. Conf. on Testing Comp. Sw.*, pages 16–18, 1999.

[16] B. Miranda. A proposal for revisiting coverage testing metrics. In *ACM/IEEE International Conference on Automated Software Engineering, ASE'14*, pages 899–902, 2014.

[17] B. Miranda and A. Bertolino. Social coverage for customized test adequacy and selection criteria. In *9th International Workshop on Automation of Software Test, AST 2014*, pages 22–28, 2014.

[18] B. Miranda and A. Bertolino. Improving test coverage measurement for reused software. In *41st Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2015*, pages 27–34, 2015.

[19] J. D. Musa. Operational profiles in software-reliability engineering. *Software, IEEE*, 10(2):14–32, 1993.

[20] A. Orso, T. Apiwattanapong, and M. J. Harrold. Leveraging field data for impact analysis and regression testing. In *Proc. ESEC/FSE-11*, pages 128–137. ACM, 2003.

[21] A. Orso, D. Liang, M. J. Harrold, and R. Lipton. Gamma system: Continuous evolution of software after deployment. In *Proc. Int. Symp. on Sw. Testing and Analysis*, ISSTA '02, pages 65–69. ACM, 2002.

[22] M. Staats, G. Gay, M. Whalen, and M. Heimdahl. On the danger of coverage directed test case generation. In *FASE'12*, pages 409–424. Springer, 2012.

[23] Y. Wei, B. Meyer, and M. Oriol. Is branch coverage a good measure of testing effectiveness? In *Emp. Sw. Eng. and Verification*, pages 194–212. Springer, 2012.

[24] W. Wong, J. Horgan, S. London, and A. Mathur. Effect of test set size and block coverage on the fault detection effectiveness. In *Proc. 5th Int. Symposium on Software Reliability Engineering*, pages 230–238, 1994.

[25] T. Xie and D. Notkin. Checking inside the black box: regression testing by comparing value spectra. *IEEE Trans. Softw. Eng.*, 31(10):869–883, Oct 2005.

[26] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM Comput. Surv.*, 29(4):366–427, Dec. 1997.