Rethinking coverage testing measures by taking into account the relevance of covered entities

Antonia Bertolino
ISTI - CNR

Breno Miranda
University of Pisa / ISTI - CNR

ICMC - São Carlos (USP) Sept. 16 2016

Abstract

The talk will introduce a novel approach to measure coverage in software testing, aimed at focusing test resources on the most “relevant” program parts. The intuitive idea is that depending on the specific testing context, reaching full coverage might not be always a meaningful target, because not all available entities are necessarily of interest in any context. With reference to some generic user-related constraints, we introduce the notion of a "testing scope" to refer to a subset of the input domain that is delimited by those constraints. Then we introduce a revised definition of test coverage, referred to as "scope-based test coverage", targetting relevant, or "in-scope", entities. In other words, we propose, as simple as it may sound, to change the denominator of the traditional coverage equation to count only those entities that are relevant in the given testing scope. Clearly, the challenge is how to properly define scope so that scope-based coverage can be automated. We have instantiated scope-based coverage in different contexts, including code reuse and reliability testing.

It's a small world

This talk is about Software Testing i.e.:

- the dynamic verification of the behavior of a program
- on a finite set of test cases
- suitably selected from the (in practice infinite) input domain
- against the expected behavior

Above is my comprehensive definition of software testing, in Software Testing ch. of the SWEBOK Guide (2001 and following editions)

Software Testing has many limits

You can never test a program exhaustively (*only exhausted things are time and money … *)

- cannot test every valid input or every execution path;
- and, even worse, cannot test every invalid input.

You can never know whether you have just found the last fault

Software Testing has many limits

Edsger W. Dijkstra (1930-2002)

"Program testing can be used to show the presence of bugs, but never to show their absence!"

Research seeks provably effective strategies and tools to overcome / mitigate software testing limits

Coverage Testing

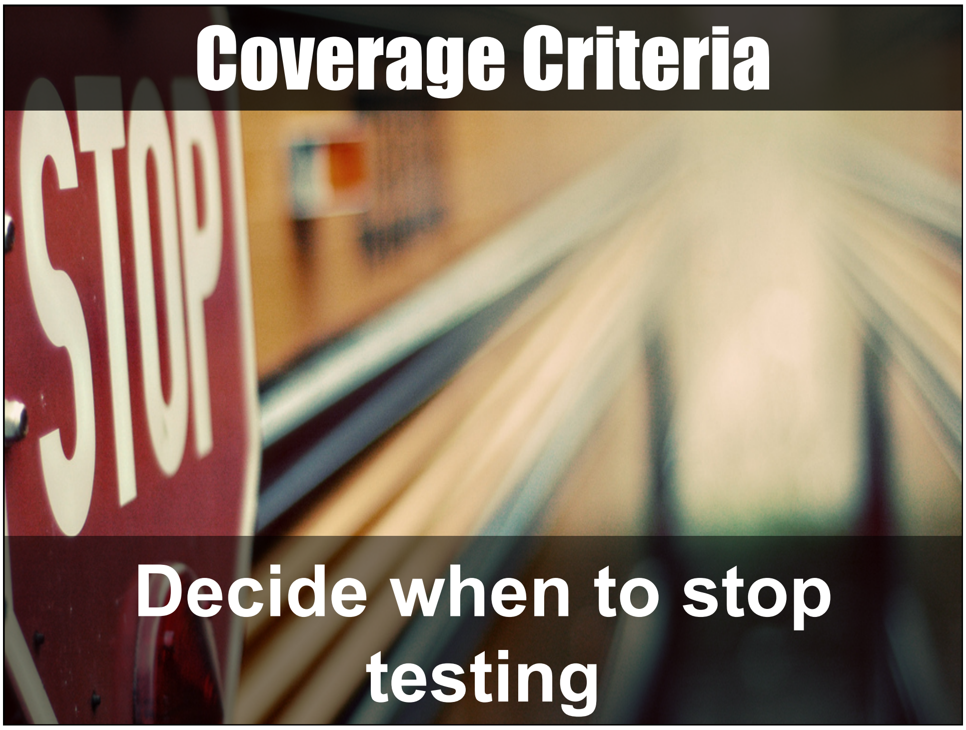
Coverage Criteria

A set of entities to be covered is defined, and a program is not considered to be adequately tested until all entities have been executed

Coverage Criteria

A set of entities to be covered is defined, and a program is not considered to be adequately tested until all entities have been executed

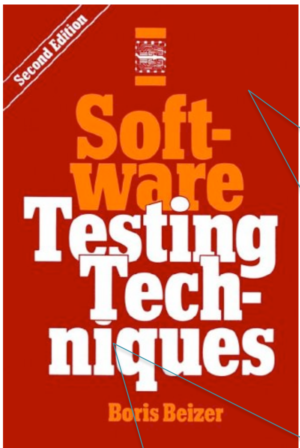
(and validated against an oracle)


Coverage Criteria
STOP
Decide when to stop testing


Coverage Criteria
SOMEWHERE ELSE
THAT WAY
THIS WAY
Guide testers in enhancing their test suites

COVERAGE MEASURE:

$$\frac{\text{\# OF COVERED ENTITIES}}{\text{\# OF AVAILABLE ENTITIES}}$$



Branch and statement coverage are accepted today as the minimum mandatory testing requirement.

In case I haven't made myself clear, leaving untested code in a system is stupid, shortsighted and irresponsible.

TRADITIONAL COVERAGE

$$\frac{\#\ \text{OF COVERED ENTITIES}}{\#\ \text{OF AVAILABLE ENTITIES}}$$

Is aiming at full coverage always meaningful?

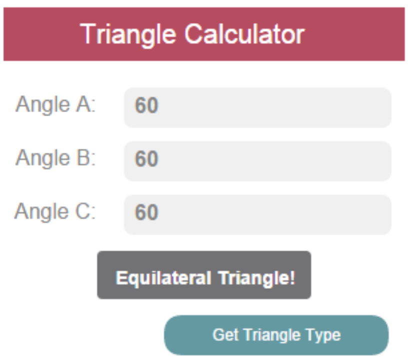
The Triangle Calculator

```
void get_triangle_type(int a, int b, int c) {
    if (a + b + c == 180) {
        if (a == b && b == c) {
            printf("Equilateral Triangle. \n");
        }
        else if (a == b || b == c || a == c) {
            printf("Isosceles Triangle. \n");
        }
        else {
            printf("Scalene Triangle. \n");
        }
    }
    else {
        printf("Triangle formation not possible\n");
    }
}

int main( int argc, char *argv[] ) {
    int a, b, c;

    sscanf(argv[1], "%d", &a);
    sscanf(argv[2], "%d", &b);
    sscanf(argv[3], "%d", &c);

    get_triangle_type(a, b, c);
}
```



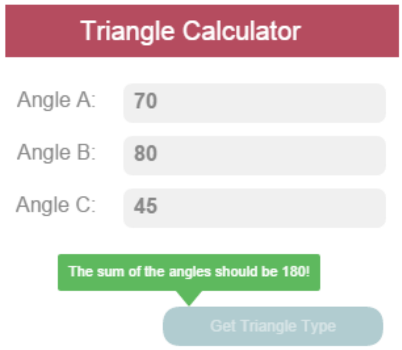
The Triangle Calculator

```c
void get_triangle_type(int a, int b, int c) {
    if (a + b + c == 180) {
        if (a == b && b == c) {
            printf("Equilateral Triangle. \n");
        }
        else if (a == b || b == c || a == c) {
            printf("Isosceles Triangle. \n");
        }
        else {
            printf("Scalene Triangle. \n");
        }
    }
    else {
        printf("Triangle formation not possible\n");
    }
}

int main( int argc, char *argv[] ) {
    int a, b, c;

    sscanf(argv[1], "%d", &a);
    sscanf(argv[2], "%d", &b);
    sscanf(argv[3], "%d", &c);

    get_triangle_type(a, b, c);
}
```




Our proposal


RELATIVE COVERAGE
OF COVERED ENTITIES
OF AVAILABLE ENTITIES

RELATIVE COVERAGE

$$\frac{\text{\# OF COVERED ENTITIES}}{\text{\# OF IN-SCOPE ENTITIES}}$$

↓

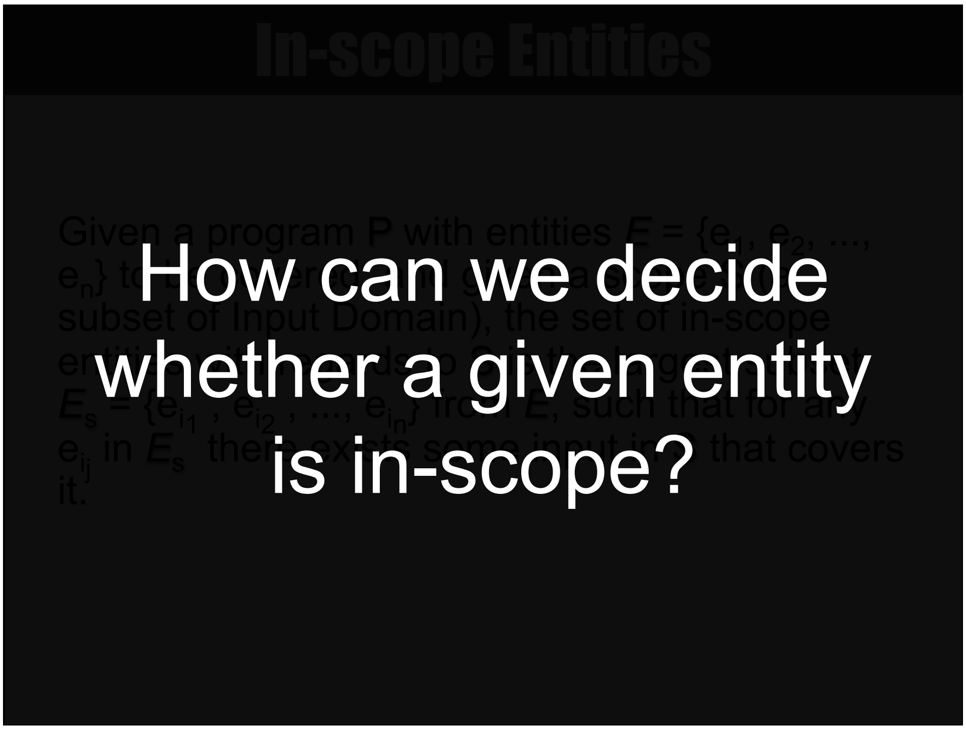
I.E., THOSE THAT ARE RELEVANT
IN THE CONTEXT OF USAGE

In-scope Entities

Definition

Given a program P with entities
$E = \{e_1, e_2, ..., e_n\}$ to be covered,
and given a scope
S (= a subset of Input Domain),
the set of in-scope entities wrt S is
the largest subset $E_s = \{e_{i_1}, e_{i_2}, ..., e_{i_n}\}$
from E, such that for any e_{i_j} in E_s
there exists some input in S that covers it

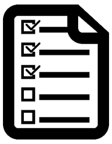

RELATIVE COVERAGE
OF COVERED ENTITIES
OF IN-SCOPE ENTITIES
Nice idea but ...


In-scope Entities
How can we decide whether a given entity is in-scope?

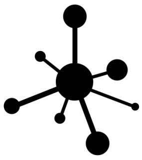
It will depend on the usage context!

- We introduced **three new adequacy criteria** inspired by the idea of relative coverage

We baptized each of them with specific names for ease of reference, but all of them are simply **different instantiations** of the **relative coverage** concept



Relevant Coverage
Code entities targeted in the context of software reuse (source code is available but cannot be changed)



Social Coverage
Operations covered by similar users, e.g., in the context of service-oriented architecture (source code is *not* available)

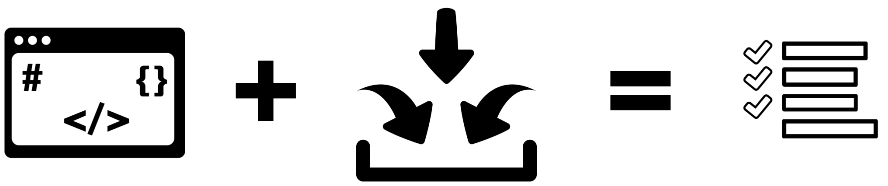
Operational Coverage
Usage profile mapped to code entities in the context of reliability testing

Relevant Coverage
Code entities targeted in the context of software reuse (source code is available but cannot be changed)

Social Coverage
Operations covered by similar users, e.g., in the context of service-oriented architecture (source code is *not* available)

Operational Coverage
Usage profile mapped to code entities in the context of reliability testing

Reuse (source code available)



Source code + **Input domain information** = **In-scope entities**

We applied Dynamic Symbolic Execution to identify those entities (specifically functions, statements, branches) that are reachable when the relevant input constraints hold

ARTICLE IN PRESS

JID: JSS [m5G;July 7, 2016;14:56]

The Journal of Systems and Software 000 (2016) 1–22

ELSEVIER

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Scope-aided test prioritization, selection and minimization for software reuse

Breno Miranda[a,b,*], Antonia Bertolino[b]

[a] Dipartimento di Informatica, Università di Pisa, Largo B. Pontecorvo, 3, 56127 Pisa, Italy
[b] ISTI - CNR, Via G. Moruzzi 1, 56124 Pisa, Italy

ARTICLE INFO

Article history:
Received 17 June 2015
Revised 11 February 2016
Accepted 21 June 2016
Available online xxx

Keywords:
In-scope entity
Test case selection
Test case prioritization
Test suite minimization
Test of reused code
Testing scope

ABSTRACT

Software reuse can improve productivity, but does not exempt developers from the need to test the reused code into the new context. For this purpose, we propose here specific approaches to white-box test prioritization, selection and minimization that take into account the reuse context when reordering or selecting test cases, by leveraging possible constraints delimiting the new input domain scope. Our scope-aided testing approach aims at detecting those faults that under such constraints would be more likely triggered in the new reuse context, and is proposed as a boost to existing approaches. Our empirical evaluation shows that in test suite prioritization we can improve the average rate of faults detected when considering faults that are in scope, while remaining competitive considering all faults; in test case selection and minimization we can considerably reduce the test suite size, with small to no extra impact on fault detection effectiveness considering both in-scope and all faults. Indeed, in minimization, we improve the in-scope fault detection effectiveness in all cases.

Prioritization (Problem)

Prioritization:

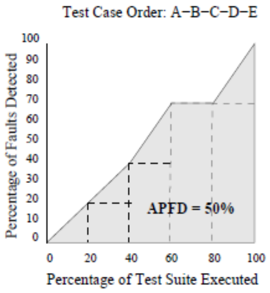
Given: *A test suite T; the set PT of permutations of T; a function f from PT to the real numbers $\mathcal{R}$*

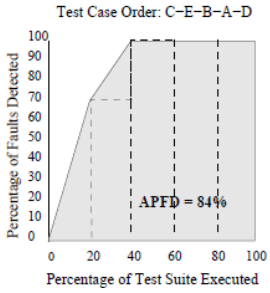
Problem: *Find $T' \in PT$ such that $\forall T''$: $(T'' \in PT)$ and $(T'' \neq T')$: $[f(T') \geq f(T'')]$*

Prioritization (Evaluation)

Average Percentage of Faults Detected (APFD)

Test	Fault 1	2	3	4	5	6	7	8	9	10
A	X				X					
B						X	X			
C	X	X	X	X	X	X	X			
D					X					
E								X	X	X





*Example from: Malishevsky, A. G., Ruthruff, J. R., Rothermel, G., & Elbaum, S. (2006). Cost-cognizant test case prioritization. *University of Nebraska-Lincoln, Techical Report*.

Selection and Minimization (Problem)

Selection:

***Given**: A program P; and a test suite T*

***Problem**: Find a subset of T, T', such that testing P with T' preserves some desired property of testing P with T*

Minimization:

***Given**: A program P; a test suite T; a set of entities $\mathcal{E} = \{e_1, ..., e_n\}$ that must be exercised to provide the desired test coverage of P; and subsets of T: $\{T_1, ..., T_n\}$, each one associated with one of the e_i such that any of the test cases $t_j \in T_i$ can be used to test e_i*

***Problem**: Find a representative set T' of test cases from T that satisfies all $e_i \in \mathcal{E}$*

Selection and Minimization (Evaluation)

Test Suite Reduction and Impact on Fault Detection Capability

$$\text{Reduction} = \left(1 - \frac{\#\ \textit{test cases in the reduced test suite}}{\#\ \textit{test cases in the original test suite}}\right) \times 100\%$$

$$\text{Impact} = \left(1 - \frac{\#\ \textit{faults detected by the reduced test suite}}{\#\ \textit{faults detected by the original test suite}}\right) \times 100\%$$

Study Subjects

- **grep** (5 versions): command-line utility that searches for lines matching a given regular expression in the provided file(s)
- **gzip** (5 versions): application used for file compression and decompression
- **sed** (7 versions): stream editor that performs basic text transformations on an input stream

Sub.	Ver.	LoC	Test Suite
grep	v1	9463	199
grep	v2	9987	199
grep	v3	10124	199
grep	v4	10143	199
grep	v5	10072	199
gzip	v1	4594	195
gzip	v2	5083	195
gzip	v3	5095	195
gzip	v4	5233	195
gzip	v5	5745	195
sed	v1	5486	360
sed	v2	9867	360
sed	v3	7146	360
sed	v4	7086	363
sed	v5	13398	370
sed	v6	13413	370
sed	v7	14456	370
	Total:	146391	4523

Testing Scopes (example)

1. It is used, within a bigger system, for **compressing files only**
2. It is used by an online service only for **decompressing** the files submitted by the service's users
3. It is used for **compressing** whole directories **recursively**

Experiment

- Applied traditional prioritization, selection, and minimization techniques on the object's test suite
- Applied our scope-aided prioritization, scope-aided selection, and scope-aided minimization on top of the traditional techniques
- Evaluated the performance of the scope-aided approach when compared to the original techniques

Prioritization Study

RQ1.1: how does scope-aided prioritization compare with original (not scope-aided) prioritization with respect to fault detection rate when considering *in-scope faults*?

RQ1.2: how does scope-aided prioritization compare with original (not scope-aided) prioritization with respect to fault detection rate when considering *all faults*?

In-scope Faults

- **In-scope fault**. A fault that may manifest itself as a failure under the scope inputs subset.

Prioritization Study

RQ1.1: Rate of Faults Detected (*in-scope faults*)

Average **APFD$_C$** (and coefficient of variation) when considering different **prioritization approaches** and different **coverage criteria**

Approach	Function		Statement		Branch	
	original	scope-aided	original	scope-aided	original	scope-aided
Total	77.0 (0.35)	**87.6** (0.14)	75.6 (0.34)	**81.0** (0.24)	74.2 (0.34)	**80.6** (0.24)
Additional	92.1 (0.07)	**92.3** (0.07)	94.1 (0.06)	**94.9** (0.05)	94.7 (0.05)	**95.5** (0.04)
Similarity	83.6 (0.18)	**87.3** (0.10)	86.1 (0.13)	**88.1** (0.08)	86.4 (0.12)	**88.5** (0.06)
Search-based	89.8 (0.08)	**90.2** (0.08)	91.6 (0.06)	90.2 (0.08)	91.6 (0.05)	90.2 (0.08)
Average:	85.7	**89.4**	86.8	**88.5**	86.7	**88.7**

Prioritization Study

RQ1.1: Rate of Faults Detected (*in-scope faults*)

Average **APFD$_C$** (and coefficient of variation) when considering different **fractions** of the prioritized suites

Coverage criterion	Fraction: 75%		Fraction: 50%		Fraction: 25%	
	original	scope-aided	original	scope-aided	original	scope-aided
Function	88.0 (0.13)	**88.6** (0.11)	87.6 (0.13)	**88.2** (0.10)	85.0 (0.13)	**85.1** (0.10)
Statement	88.9 (0.11)	88.8 (0.12)	86.2 (0.13)	**87.9** (0.10)	84.4 (0.13)	**86.2** (0.09)
Branch	89.0 (0.10)	88.3 (0.13)	87.9 (0.10)	87.3 (0.12)	85.1 (0.10)	**86.4** (0.09)
Average:	88.6	88.6	87.2	**87.8**	84.8	**85.9**

Prioritization Study

RQ1.2: Rate of Faults Detected (*all faults*)

Average **APFD$_C$** (and coefficient of variation) when considering different **prioritization approaches** and different **coverage criteria**

Approach	Function		Statement		Branch	
	original	scope-aided	original	scope-aided	original	scope-aided
Total	74.4 (0.24)	**78.4** (0.23)	73.8 (0.23)	**76.0** (0.26)	70.6 (0.31)	**74.7** (0.30)
Additional	92.9 (0.07)	92.6 (0.07)	95.5 (0.05)	95.4 (0.04)	96.2 (0.04)	95.9 (0.03)
Similarity	84.3 (0.11)	**86.5** (0.09)	85.9 (0.08)	85.7 (0.10)	87.0 (0.06)	86.5 (0.10)
Search-based	91.5 (0.04)	**91.6** (0.04)	93.1 (0.04)	91.6 (0.04)	92.8 (0.03)	91.5 (0.03)
Average:	85.8	**87.3**	87.1	**87.2**	86.6	**87.1**

Minimization Study

RQ3.1: *Test suite reduction*: how does scope-aided minimization compare with the original one (not scope-aided) in terms of test suite reduction achieved?

RQ3.2: *Impact on fault detection capability*: what is the impact of scope-aided minimization with respect to the test suite's fault detection capability when compared to the original (not scope-aided) minimization and considering both *all faults* and *in-scope faults*?

Minimization Study

RQ3.1: Test Suite Reduction

Average **test suite reduction** (and coefficient of variation) achieved by the **scope-aided minimization** and the **traditional approach**

Subject versions	*grep*		*gzip*		*sed*	
	original	scope-aided	original	scope-aided	original	scope-aided
V1	77.7% (0.17)	**87.4%** (0.11)	91.6% (0.02)	**97.3%** (0.01)	94.1% (0.04)	**97.5%** (0.02)
V2	77.7% (0.17)	**88.9%** (0.09)	91.8% (0.02)	**97.1%** (0.02)	93.9% (0.04)	**98.1%** (0.02)
V3	78.4% (0.16)	**88.9%** (0.10)	91.8% (0.02)	**97.5%** (0.01)	93.8% (0.03)	**97.3%** (0.02)
V4	78.6% (0.16)	**89.2%** (0.09)	91.8% (0.02)	**97.4%** (0.01)	93.4% (0.04)	**97.3%** (0.02)
V5	78.6% (0.16)	**89.3%** (0.09)	91.8% (0.03)	**97.3%** (0.02)	93.8% (0.04)	**97.2%** (0.02)
V6	-	-	-	-	93.8% (0.04)	**96.6%** (0.03)
V7	-	-	-	-	93.3% (0.04)	**97.8%** (0.01)
Average:	78.2%	**88.7%**	91.8%	**97.3%**	93.7%	**97.4%**

Minimization Study

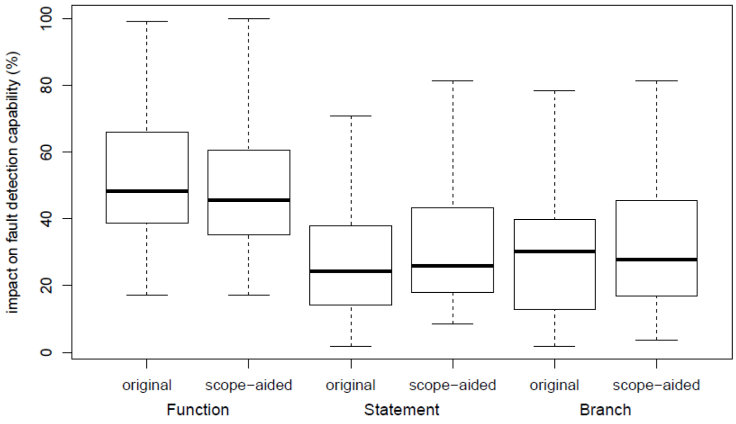
RQ3.2: Impact on Fault Detection Capability

$$\text{Impact} = \left(1 - \frac{\#\ faults\ detected\ by\ the\ reduced\ test\ suite}{\#\ faults\ detected\ by\ the\ original\ test\ suite}\right) \times 100\%$$

Minimization Study

RQ3.2: Impact on Fault Detection Capability

Impact on fault detection capability for the different coverage criteria when considering the set of *all faults*

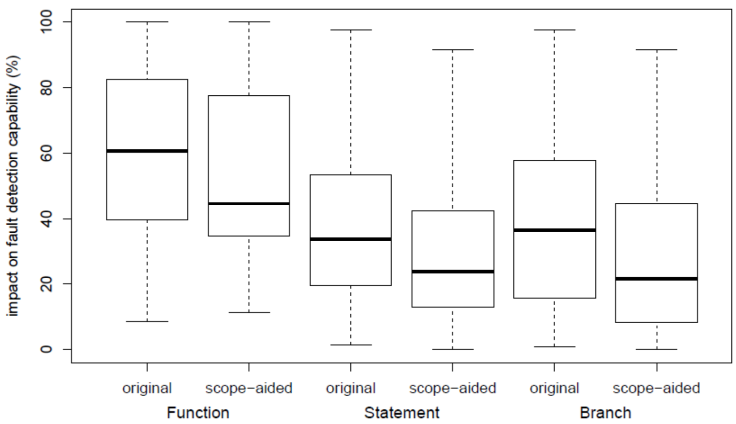


Wilcoxon signed-rank test: median differences are statistically significant, at 5% level, for statement (***p* = 0.01392**) and branch (***p* = 0.04964**), but not for function (***p* = 0.4014**).

Minimization Study

RQ3.2: Impact on Fault Detection Capability

Impact on fault detection capability for the different coverage criteria when considering the set of *in-scope faults*



Wilcoxon signed-rank test: the difference in the median values is statistically significant, at 5% level, for branch (***p* = 0.03909**), but not for function (***p* = 0.05116**) and statement (***p* = 0.1595**).

In summary, scope-aided approach:

- For **prioritization**:
  - Used as a burst to total and additional greedy heuristics; to similarity-based approach; and to one search-based technique
  - Found the **most important faults faster**
- For **selection** and **minimization**:
  - Compared with greedy approaches
  - **Reduced the test suite size** while **maintaining comparable fault detection capability**

Relevant Coverage
Code entities targeted in the context of software reuse (source code is available but cannot be changed)

Social Coverage
Operations covered by similar users, e.g., in the context of service-oriented architecture (source code is *not* available)

Operational Coverage
Usage profile mapped to code entities in the context of reliability testing

Software Reliability

John D. Musa
(1933-2009)

Operational Profile: *a quantitative characterization of how a system will be used.*

*"A software-based product's reliability depends on just **how a customer will use it**. Making a good reliability estimate depends on testing the product as if it were in the field"* [1]

[1] J. D. Musa. *Operational profiles in software-reliability engineering*. IEEE Software 10:14-32, 1993.

Operational Profile Based Testing

Motivating Scenario

Operational Profiles for a Publication Management System

Operations	Occurrence Probability		
	Authors	Librarians	System Administrators
Add publication	0.20	0.15	0.0
Browse publications	0.70	0.38	0.0
Add users	0.0	0.15	0.20
Remove users	0.0	0.06	0.10
…	…	…	…
Set/Update user permissions	0.0	0.06	0.21
Database backup	0.0	0.06	0.42

Operational Profile Based Testing

Motivating Scenario

The system fulfills all my needs!

Operational Profiles for a … nagement System

Operations	… Probability		
	Authors	Librarians	System Administrators
Add publication	0.20	0.15	0.0
Browse publications	0.70	0.38	0.0
Add users	0.0	0.15	0.20
Remove users	0.0	0.06	0.10
…	…	…	…
Set/Update user permissions	0.0	0.06	0.21
Database backup	0.0	0.06	0.42

Operational Profile Based Testing

Motivating Scenario

It is fairly reliable!

Operational Profiles for a Public… nt System

Operations	Occu… y		
	Authors	**Librarians**	System Administrators
Add publication	0.20	0.15	0.0
Browse publications	0.70	0.38	0.0
Add users	0.0	0.15	0.20
Remove users	0.0	0.06	0.10
…	…	…	…
Set/Update user permissions	0.0	0.06	0.21
Database backup	0.0	0.06	0.42

Operational Profile Based Testing

Motivating Scenario

Operational Profiles for a Publicat … System

… I have a different opinion!

Operations	Authors	Librarians	System Administrators
		Occurre…	
Add publication	0.20	0.15	0.0
Browse publications	0.70	0.38	0.0
Add users	0.0	0.15	0.20
Remove users	0.0	0.06	0.10
…	…	…	…
Set/Update user permissions	0.0	0.06	0.21
Database backup	0.0	0.06	0.42

Coverage Metrics

$$Traditional\ Coverage = \#\ covered\ entities / \#\ available\ entities$$

$$Relative\ Coverage = \#\ covered\ entities / \#\ in\text{–}scope\ entities$$

Hit Spectrum

Branch ID	Hit
1	1
2	1
3	1
4	0
5	1
6	1
7	1
8	1
9	0
10	1
11	1
12	1
13	0
14	1
15	1

Hit Spectrum

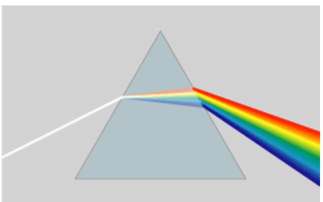
Branch ID	Hit
1	1
2	1
3	1
4	0
5	1
6	1
7	1
8	1
9	0
10	1
11	1
12	1
13	0
14	1

Does not capture entities frequency

Hit Spectrum x Count Spectrum

Branch ID	Hit	Count
1	1	4278
2	1	10834
3	1	11623
4	0	0
5	1	4876
6	1	3972
7	1	10543
8	1	2187
9	0	0
10	1	2267
11	1	2087
12	1	1678
13	0	0
14	1	5458
15	1	9876

Operational profile based testing



We introduce coverage measures based on **program count spectra:** i.e., in addition to distinguishing between in-scope and out-of-scope entities, we also take into account how much in-scope entities are exercised

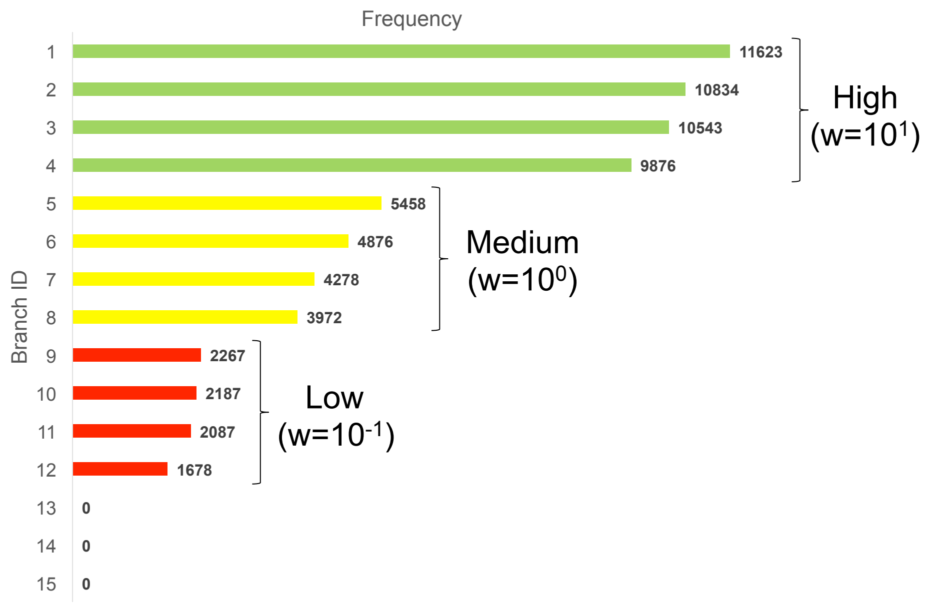
- A program count spectrum rates entities based on their **usage frequency**.

We considered:

- Branch-count spectrum (BCS)
- Statement-count spectrum (SCS)
- Function-count spectrum (FCS)

And for each case we clustered entities into 3 groups: High, Medium and Low

Importance Groups



Operational Coverage

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

where:

n = number of importance groups
x_i = the rate of covered entities from group i
w_i = the weight assigned to group i

Research Questions

RQ1: Does operational coverage provide a good stopping rule (*adequacy criterion*) for operational profile based testing?

RQ2: Is operational coverage useful for selecting test cases (*selection criterion*) for operational profile based testing?

Study Subjects

Subject	Version	LoC	# Seeded faults
grep	V3	10124	18
gzip	V4	5233	12
sed	V2	9867	5
	Total:	**25224**	**35**

Adequacy Study

Tasks and Procedures

- Carry out operational profile based testing by selecting the next test case to be run according to the occurrence probabilities defined in the customized operational profile
- After each test case is run, we calculate:
  1. Traditional coverage
  2. Operational coverage
  3. The probability of failure for the next test case

Adequacy Study Results

RQ1: correlation between coverage and failure probability

Kendall Tau correlation between coverage and the probability that the next test case will not fail (all entries significant at 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**

Correlation	Guildford scale [2]
< 0.4	"low"
>= 0.4 and < 0.7	"moderate"
>= 0.7 and < 0.9	"high"
>= 0.9	"very high"

[2] Joy Paul Guilford, *Fundamental statistics in psychology and education*. McGraw-Hill, 1942.

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.

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 effec-

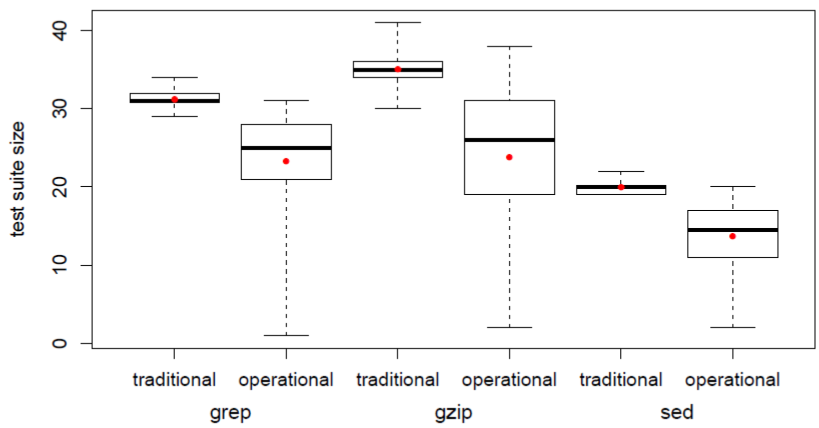
Selection Study

Tasks and Procedures

- Derive two test suites using the *greedy additional* heuristic:
  - The first test suite targets all the entities available in the subject under testing. We refer to it as the traditional test suite.
  - The second one, the operational test suite, targets the most important entities for the customized operational profile.
- We then measure:
  - The size of the derived test suites
  - The remaining failure probability

Selection Study Results

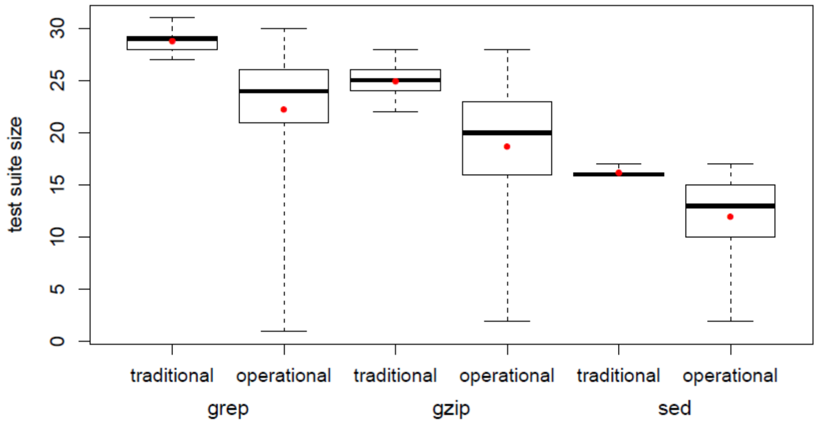
Test suite reduction (Branch coverage)



Wilcoxon signed-rank test: all the median differences are statistically significant at the 5% level.
p-value **< 2.2e−16**

Selection Study Results

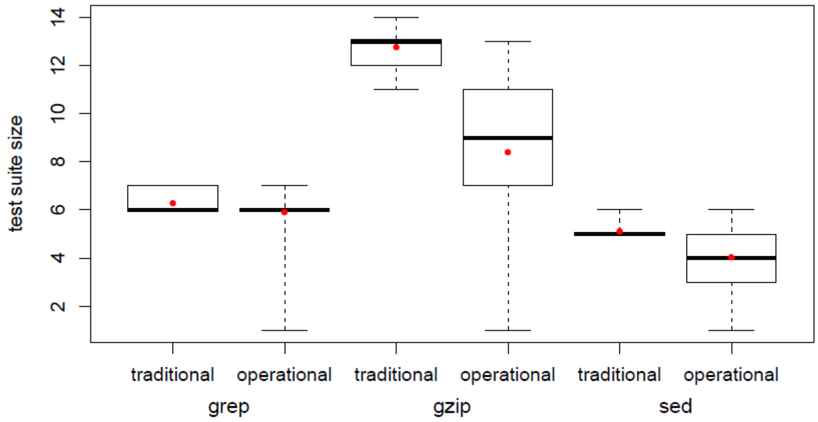
Test suite reduction (Statement coverage)



Wilcoxon signed-rank test: all the median differences are statistically significant at the 5% level.
p-value **< 2.2e−16**

Selection Study Results

Test suite reduction (Function coverage)



Wilcoxon signed-rank test: all the median differences are statistically significant at the 5% level.
p-value **< 2.2e−16**

Selection Study Results

RQ2: Remaining failure probability after test suite execution

Subject	Branch		Statement		Function	
	trad.	*oper.*	*trad.*	*oper.*	*trad.*	*oper.*
grep	2.720	**0.907**	2.180	**0.804**	7.113	7.815
gzip	0.003	0.063	0.056	**0.043**	1.200	**0.966**
sed	0.205	**0.147**	0.306	**0.174**	15.125	**13.682**
Average:	0.976	**0.372**	0.847	**0.340**	7.813	**7.488**

Wilcoxon signed-rank test: all the median differences are statistically significant at the 5% level.

In summary:

- We defined a **coverage criterion** for **operational profile based testing**
- We proposed a novel method of **measuring code coverage** that exploits **program spectra**
- We conducted **the first study** of using operational coverage for test **adequacy** and **selection** in the context of operational profile based testing

Conclusions

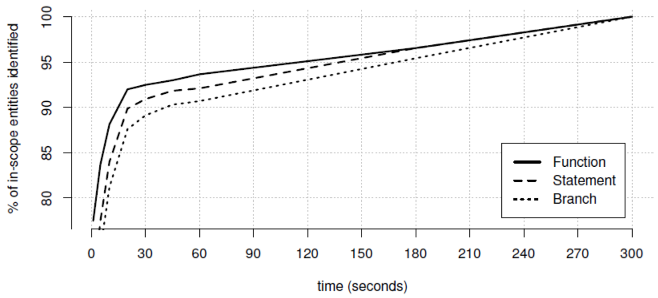
- This talk aimed at demonstrating the very idea of "relative coverage"
- The final goal would be –given a test context- a fully automated solution from user's constraints all way down to relative coverage testing

To keep in mind…

- Relative coverage should **not** be taken as **an alternative metric** for the purpose of achieving a **higher coverage score**
- Also, it should not to be taken as an advice to test **"less"**
- Good for **reliability**, not for **safety**!

Conclusions (On the costs)

- The only extra cost added is related to the **identification of the in-scope entities**
  – It will depend on the method chosen



Average % of **in-scope entities** identified over time for ***grep***, ***gzip***, and ***sed*** when using **dynamic symbolic exploration** supported by **KLEE**

Future work

- A practical approach, which needs practitioners' feedback
- While we have developed some instantiations, the notion is general and can be applied in varying contexts
- We would be highly interested in evaluating the approach with an industrial partner

Future Work

- Investigate **different approaches** for the **identification of the in-scope entities**
- Investigate the impact of the in-scope entities on **test case generation**
  - *How effective would be a test suite generated targeting the set of in-scope entities?*

THANKS!