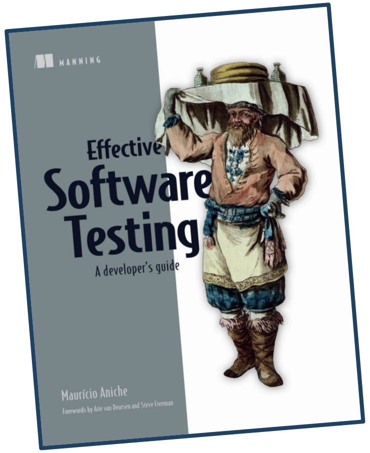
How coverage can be used (and abused) to guide testing

Mauricio Aniche

https://www.effective-software-testing.com

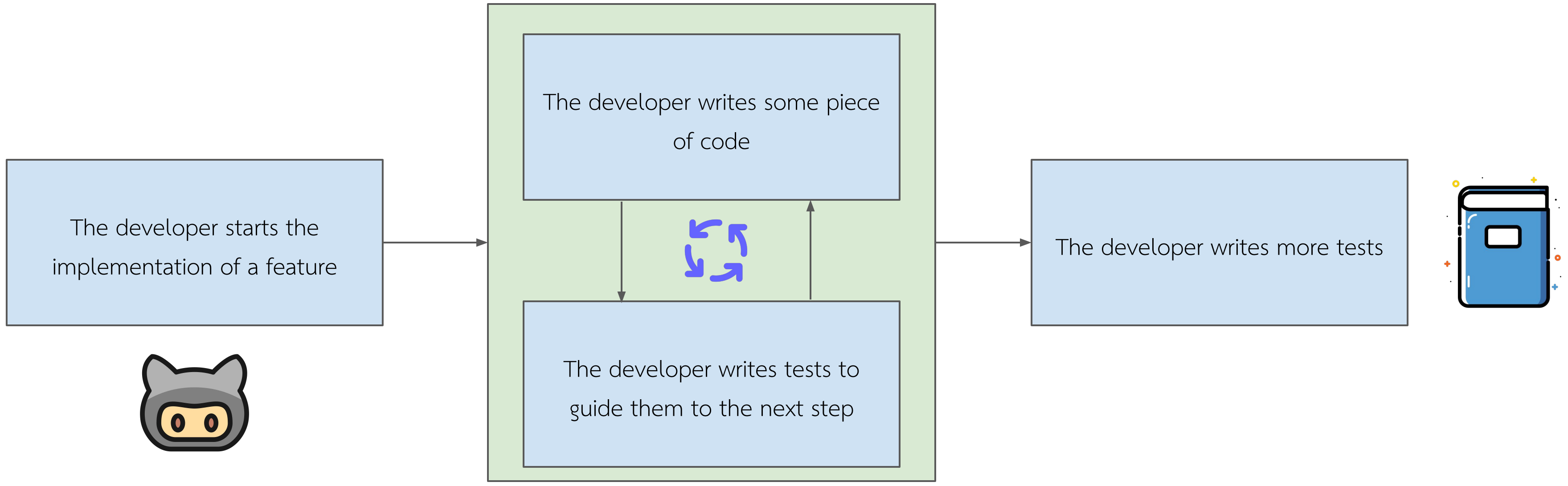

MANNING
Effective Software Testing
A developer's guide
Maurício Aniche
Forewords by Arie van Deursen and Steve Freeman

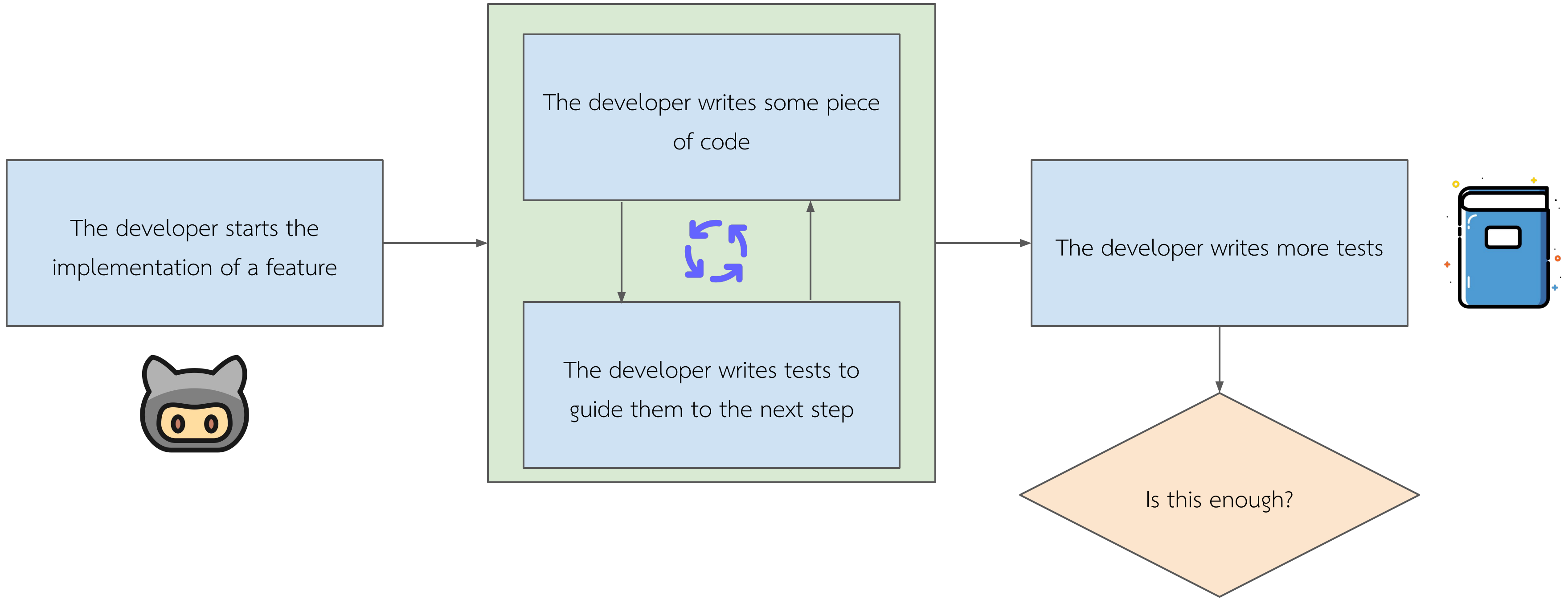
Cove coverage is useless because ...

- It can be easily tricked!
- It forces me to write useless tests, like tests for my getters!
- I can have tests without assertions, and coverage will be high!
- 100% coverage doesn't mean your tests are good!

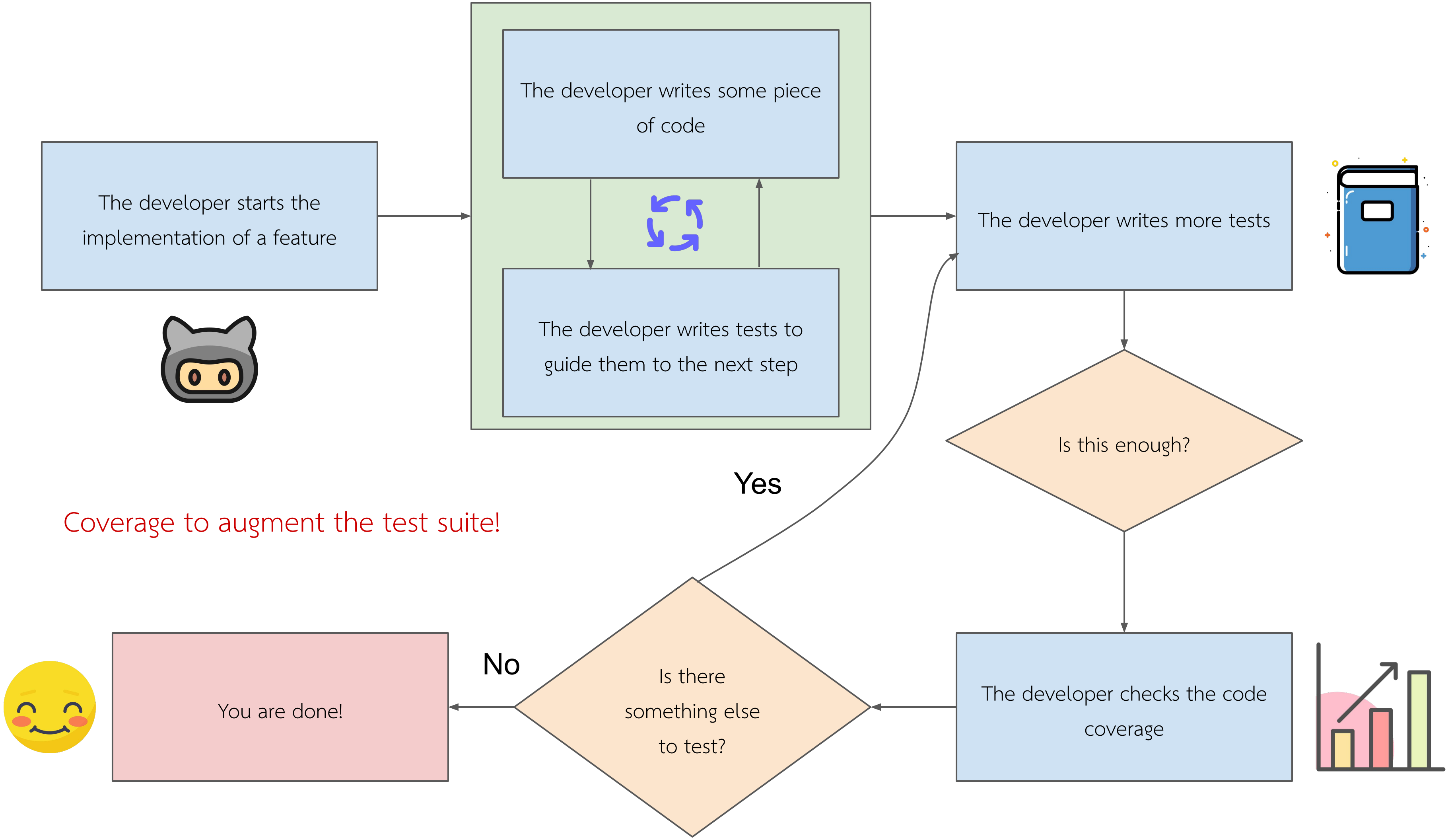
If you hate code coverage, that's because you **are not** using it properly!

Coverage should be used **to augment your test suite**, and not as something you gotta have ...!


The developer starts the implementation of a feature
The developer writes some piece of code
The developer writes tests to guide them to the next step
The developer writes more tests


The developer starts the implementation of a feature
The developer writes some piece of code
The developer writes tests to guide them to the next step
The developer writes more tests
Is this enough?

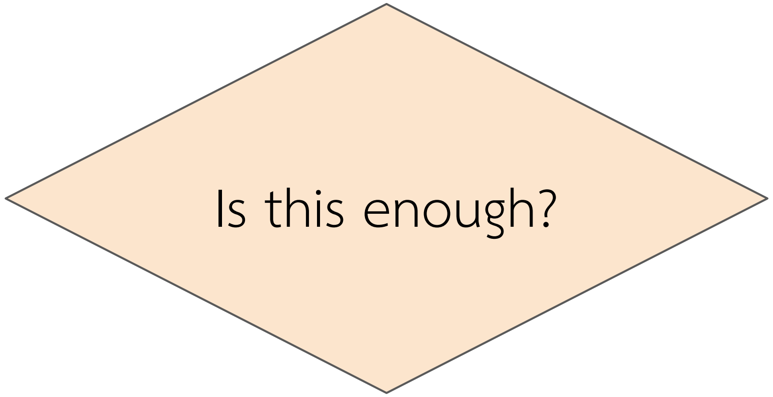
What's where coverage steps in!


The developer starts the implementation of a feature
The developer writes some piece of code
The developer writes tests to guide them to the next step
The developer writes more tests
Is this enough?
The developer checks the code coverage
Is there something else to test?
Yes
No
You are done!

Coverage to augment the test suite!

How to take the "we are done" decision?

- Why is it not covered by a test?
    - Should we cover it?
- Does this test have a real chance of revealing a bug or a regression?
- ... or am I just doing it to increase my coverage number?



Code coverage **informs the decision**, but it shouldn't take the decision for you!

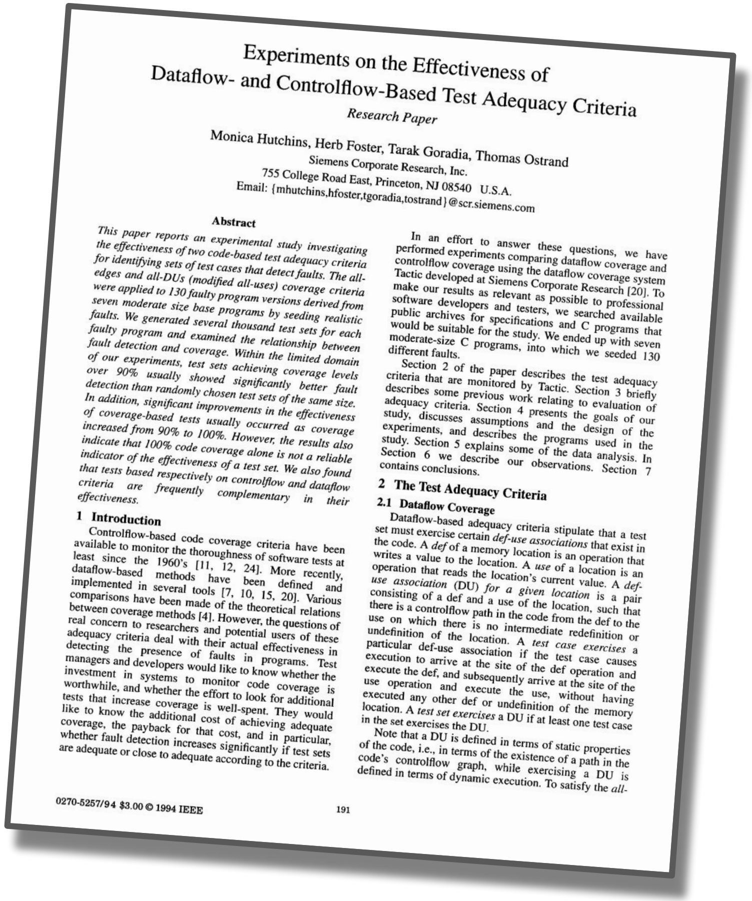
Does higher coverage really

lead to better

software?

- Test suites that have over 90% coverage do better at detecting faults!
- 100% code coverage alone is not a reliable indicator of the effectiveness of a test set.

— Hutchins et al. (1994)



Experiments on the Effectiveness of
Dataflow- and Controlflow-Based Test Adequacy Criteria

Research Paper

Monica Hutchins, Herb Foster, Tarak Goradia, Thomas Ostrand
Siemens Corporate Research, Inc.
755 College Road East, Princeton, NJ 08540 U.S.A.
Email: {mhutchins,hfoster,tgoradia,tostrand}@scr.siemens.com

Abstract

This paper reports an experimental study investigating the effectiveness of two code-based test adequacy criteria for identifying sets of test cases that detect faults. The all-edges and all-DUs (modified all-uses) coverage criteria were applied to 130 faulty program versions derived from seven moderate size base programs by seeding realistic faults. We generated several thousand test sets for each faulty program and examined the relationship between fault detection and coverage. Within the limited domain of our experiments, test sets achieving coverage levels over 90% usually showed significantly better fault detection than randomly chosen test sets of the same size. In addition, significant improvements in the effectiveness of coverage-based tests usually occurred as coverage increased from 90% to 100%. However, the results also indicate that 100% code coverage alone is not a reliable indicator of the effectiveness of a test set. We also found that tests based respectively on controlflow and dataflow criteria are frequently complementary in their effectiveness.

1 Introduction

Controlflow-based code coverage criteria have been available to monitor the thoroughness of software tests at least since the 1960's [11, 12, 24]. More recently, dataflow-based methods have been defined and implemented in several tools [7, 10, 15, 20]. Various comparisons have been made of the theoretical relations between coverage methods [4]. However, the questions of real concern to researchers and potential users of these adequacy criteria deal with their actual effectiveness in detecting the presence of faults in programs. Test managers and developers would like to know whether the investment in systems to monitor code coverage is worthwhile, and whether the effort to look for additional tests that increase coverage is well-spent. They would like to know the additional cost of achieving adequate coverage, the payback for that cost, and in particular, whether fault detection increases significantly if test sets are adequate or close to adequate according to the criteria.

In an effort to answer these questions, we have performed experiments comparing dataflow coverage and controlflow coverage using the dataflow coverage system Tactic developed at Siemens Corporate Research [20]. To make our results as relevant as possible to professional software developers and testers, we searched available public archives for specifications and C programs that would be suitable for the study. We ended up with seven moderate-size C programs, into which we seeded 130 different faults.

Section 2 of the paper describes the test adequacy criteria that are monitored by Tactic. Section 3 briefly describes some previous work relating to evaluation of adequacy criteria. Section 4 presents the goals of our study, discusses assumptions and the design of the experiments, and describes the programs used in the study. Section 5 explains some of the data analysis. In Section 6 we describe our observations. Section 7 contains conclusions.

2 The Test Adequacy Criteria

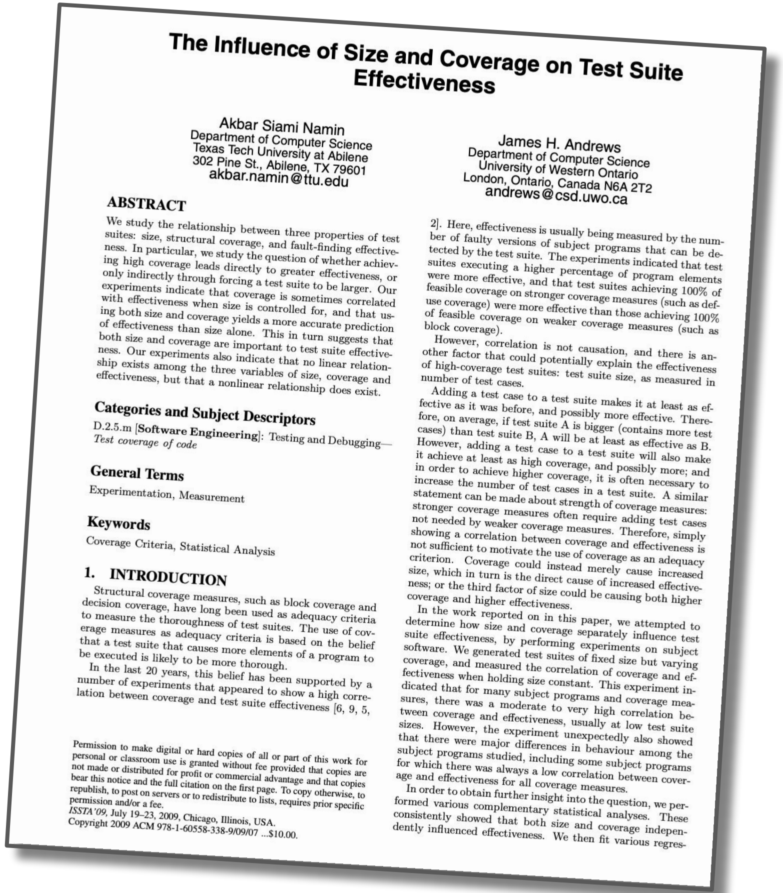
2.1 Dataflow Coverage

Dataflow-based adequacy criteria stipulate that a test set must exercise certain *def-use associations* that exist in the code. A *def* of a memory location is an operation that writes a value to the location. A *use* of a location is an operation that reads the location's current value. A *def-use association* (DU) *for a given location* is a pair consisting of a def and a use of the location, such that there is a controlflow path in the code from the def to the use on which there is no intermediate redefinition or undefinition of the location. A *test case exercises* a particular def-use association if the test case causes execution to arrive at the site of the def operation and execute the def, and subsequently arrive at the site of the use operation and execute the use, without having executed any other def or undefinition of the memory location. A *test set exercises* a DU if at least one test case in the set exercises the DU.

Note that a DU is defined in terms of static properties of the code, i.e., in terms of the existence of a path in the code's controlflow graph, while exercising a DU is defined in terms of dynamic execution. To satisfy the *all-*

- Keep adding tests by itself is not an efficient strategy!
  - Tests that cover new things are more likely to find new faults!
- Recommends the same thing I did two minutes ago: test first based on the specs and then use coverage to augment!

— Namin and Andrews (2009)



The Influence of Size and Coverage on Test Suite Effectiveness

Akbar Siami Namin
Department of Computer Science
Texas Tech University at Abilene
302 Pine St., Abilene, TX 79601
akbar.namin@ttu.edu

James H. Andrews
Department of Computer Science
University of Western Ontario
London, Ontario, Canada N6A 2T2
andrews@csd.uwo.ca

ABSTRACT

We study the relationship between three properties of test suites: size, structural coverage, and fault-finding effectiveness. In particular, we study the question of whether achieving high coverage leads directly to greater effectiveness, or only indirectly through forcing a test suite to be larger. Our experiments indicate that coverage is sometimes correlated with effectiveness when size is controlled for, and that using both size and coverage yields a more accurate prediction of effectiveness than size alone. This in turn suggests that both size and coverage are important to test suite effectiveness. Our experiments also indicate that no linear relationship exists among the three variables of size, coverage and effectiveness, but that a nonlinear relationship does exist.

Categories and Subject Descriptors

D.2.5.m [**Software Engineering**]: Testing and Debugging—*Test coverage of code*

General Terms

Experimentation, Measurement

Keywords

Coverage Criteria, Statistical Analysis

1. INTRODUCTION

Structural coverage measures, such as block coverage and decision coverage, have long been used as adequacy criteria to measure the thoroughness of test suites. The use of coverage measures as adequacy criteria is based on the belief that a test suite that causes more elements of a program to be executed is likely to be more thorough.

In the last 20 years, this belief has been supported by a number of experiments that appeared to show a high correlation between coverage and test suite effectiveness [6, 9, 5, 2]. Here, effectiveness is usually being measured by the number of faulty versions of subject programs that can be detected by the test suite. The experiments indicated that test suites executing a higher percentage of program elements were more effective, and that test suites achieving 100% of feasible coverage on stronger coverage measures (such as def-use coverage) were more effective than those achieving 100% of feasible coverage on weaker coverage measures (such as block coverage).

However, correlation is not causation, and there is another factor that could potentially explain the effectiveness of high-coverage test suites: test suite size, as measured in number of test cases.

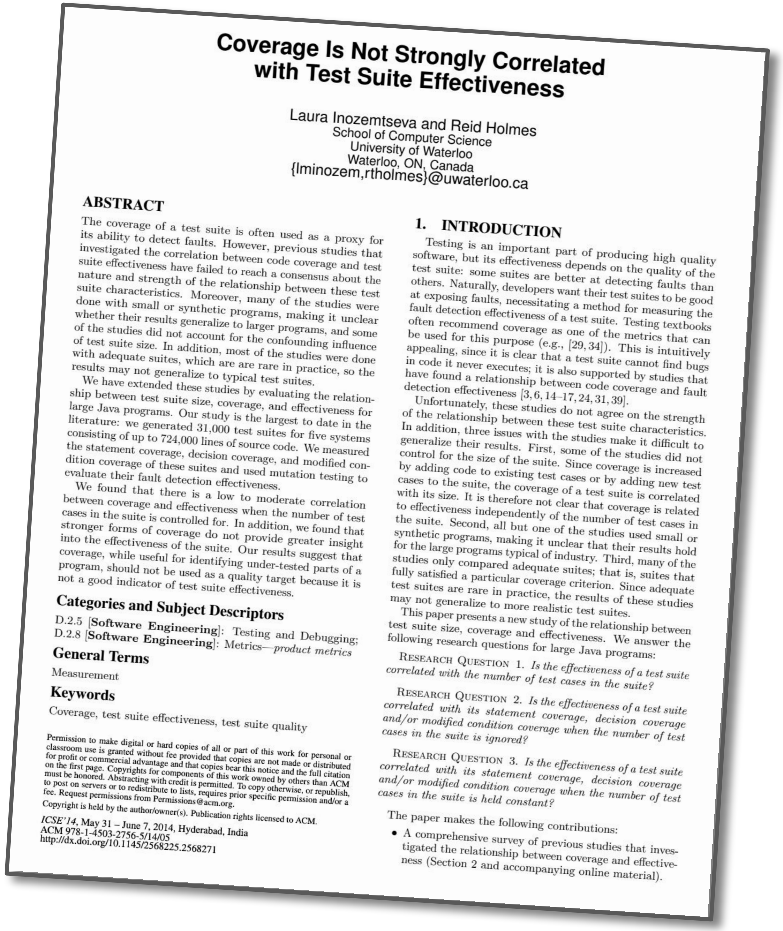
Adding a test case to a test suite makes it at least as effective as it was before, and possibly more effective. Therefore, on average, if test suite A is bigger (contains more test cases) than test suite B, A will be at least as effective as B. However, adding a test case to a test suite will also make it achieve at least as high coverage, and possibly more; and in order to achieve higher coverage, it is often necessary to increase the number of test cases in a test suite. A similar statement can be made about strength of coverage measures: stronger coverage measures often require adding test cases not needed by weaker coverage measures. Therefore, simply showing a correlation between coverage and effectiveness is not sufficient to motivate the use of coverage as an adequacy criterion. Coverage could instead merely cause increased size, which in turn is the direct cause of increased effectiveness; or the third factor of size could be causing both higher coverage and higher effectiveness.

In the work reported on in this paper, we attempted to determine how size and coverage separately influence test suite effectiveness, by performing experiments on subject software. We generated test suites of fixed size but varying coverage, and measured the correlation of coverage and effectiveness when holding size constant. This experiment indicated that for many subject programs and coverage measures, there was a moderate to very high correlation between coverage and effectiveness, usually at low test suite sizes. However, the experiment unexpectedly also showed that there were major differences in behaviour among the subject programs studied, including some subject programs for which there was always a low correlation between coverage and effectiveness for all coverage measures.

In order to obtain further insight into the question, we performed various complementary statistical analyses. These consistently showed that both size and coverage independently influenced effectiveness. We then fit various regres-

- Coverage is not always related to effectiveness.
- Good for identifying under-tested parts of the system, bad if used as a quality target!

— Inozemtseva and Holmes (2014)



Coverage Is Not Strongly Correlated with Test Suite Effectiveness

Laura Inozemtseva and Reid Holmes
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{lminozem,rtholmes}@uwaterloo.ca

ABSTRACT

The coverage of a test suite is often used as a proxy for its ability to detect faults. However, previous studies that investigated the correlation between code coverage and test suite effectiveness have failed to reach a consensus about the nature and strength of the relationship between these test suite characteristics. Moreover, many of the studies were done with small or synthetic programs, making it unclear whether their results generalize to larger programs, and some of the studies did not account for the confounding influence of test suite size. In addition, most of the studies were done with adequate suites, which are are rare in practice, so the results may not generalize to typical test suites.

We have extended these studies by evaluating the relationship between test suite size, coverage, and effectiveness for large Java programs. Our study is the largest to date in the literature: we generated 31,000 test suites for five systems consisting of up to 724,000 lines of source code. We measured the statement coverage, decision coverage, and modified condition coverage of these suites and used mutation testing to evaluate their fault detection effectiveness.

We found that there is a low to moderate correlation between coverage and effectiveness when the number of test cases in the suite is controlled for. In addition, we found that stronger forms of coverage do not provide greater insight into the effectiveness of the suite. Our results suggest that coverage, while useful for identifying under-tested parts of a program, should not be used as a quality target because it is not a good indicator of test suite effectiveness.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging; D.2.8 [**Software Engineering**]: Metrics—*product metrics*

General Terms

Measurement

Keywords

Coverage, test suite effectiveness, test suite quality

ICSE'14, May 31 – June 7, 2014, Hyderabad, India
ACM 978-1-4503-2756-5/14/05
http://dx.doi.org/10.1145/2568225.2568271

1. INTRODUCTION

Testing is an important part of producing high quality software, but its effectiveness depends on the quality of the test suite: some suites are better at detecting faults than others. Naturally, developers want their test suites to be good at exposing faults, necessitating a method for measuring the fault detection effectiveness of a test suite. Testing textbooks often recommend coverage as one of the metrics that can be used for this purpose (e.g., [29, 34]). This is intuitively appealing, since it is clear that a test suite cannot find bugs in code it never executes; it is also supported by studies that have found a relationship between code coverage and fault detection effectiveness [3, 6, 14–17, 24, 31, 39].

Unfortunately, these studies do not agree on the strength of the relationship between these test suite characteristics. In addition, three issues with the studies make it difficult to generalize their results. First, some of the studies did not control for the size of the suite. Since coverage is increased by adding code to existing test cases or by adding new test cases to the suite, the coverage of a test suite is correlated with its size. It is therefore not clear that coverage is related to effectiveness independently of the number of test cases in the suite. Second, all but one of the studies used small or synthetic programs, making it unclear that their results hold for the large programs typical of industry. Third, many of the studies only compared adequate suites; that is, suites that fully satisfied a particular coverage criterion. Since adequate test suites are rare in practice, the results of these studies may not generalize to more realistic test suites.

This paper presents a new study of the relationship between test suite size, coverage and effectiveness. We answer the following research questions for large Java programs:

RESEARCH QUESTION 1. *Is the effectiveness of a test suite correlated with the number of test cases in the suite?*

RESEARCH QUESTION 2. *Is the effectiveness of a test suite correlated with its statement coverage, decision coverage and/or modified condition coverage when the number of test cases in the suite is ignored?*

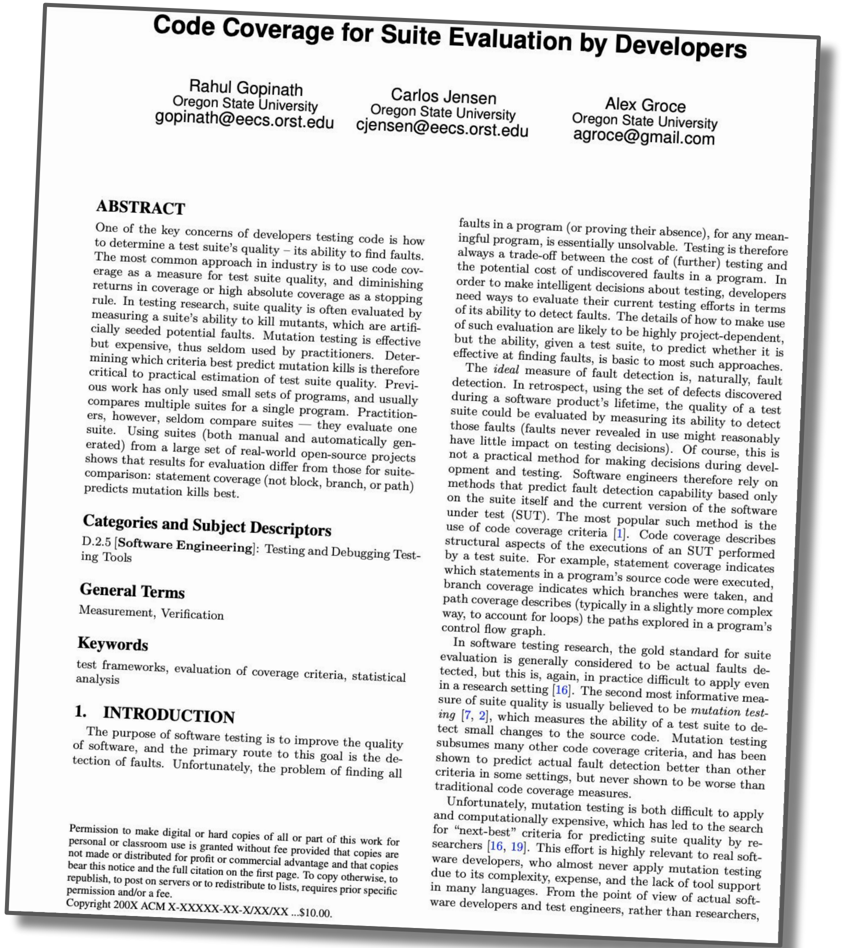
RESEARCH QUESTION 3. *Is the effectiveness of a test suite correlated with its statement coverage, decision coverage and/or modified condition coverage when the number of test cases in the suite is held constant?*

The paper makes the following contributions:

- A comprehensive survey of previous studies that investigated the relationship between coverage and effectiveness (Section 2 and accompanying online material).

- You may not need fancy coverage criteria.
- Statement coverage is already very good!

— Gopinath et al. (2014)



Code Coverage for Suite Evaluation by Developers

Rahul Gopinath
Oregon State University
gopinath@eecs.orst.edu

Carlos Jensen
Oregon State University
cjensen@eecs.orst.edu

Alex Groce
Oregon State University
agroce@gmail.com

ABSTRACT

One of the key concerns of developers testing code is how to determine a test suite's quality – its ability to find faults. The most common approach in industry is to use code coverage as a measure for test suite quality, and diminishing returns in coverage or high absolute coverage as a stopping rule. In testing research, suite quality is often evaluated by measuring a suite's ability to kill mutants, which are artificially seeded potential faults. Mutation testing is effective but expensive, thus seldom used by practitioners. Determining which criteria best predict mutation kills is therefore critical to practical estimation of test suite quality. Previous work has only used small sets of programs, and usually compares multiple suites for a single program. Practitioners, however, seldom compare suites — they evaluate one suite. Using suites (both manual and automatically generated) from a large set of real-world open-source projects shows that results for evaluation differ from those for suite-comparison: statement coverage (not block, branch, or path) predicts mutation kills best.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging Testing Tools

General Terms

Measurement, Verification

Keywords

test frameworks, evaluation of coverage criteria, statistical analysis

1. INTRODUCTION

The purpose of software testing is to improve the quality of software, and the primary route to this goal is the detection of faults. Unfortunately, the problem of finding all faults in a program (or proving their absence), for any meaningful program, is essentially unsolvable. Testing is therefore always a trade-off between the cost of (further) testing and the potential cost of undiscovered faults in a program. In order to make intelligent decisions about testing, developers need ways to evaluate their current testing efforts in terms of its ability to detect faults. The details of how to make use of such evaluation are likely to be highly project-dependent, but the ability, given a test suite, to predict whether it is effective at finding faults, is basic to most such approaches.

The *ideal* measure of fault detection is, naturally, fault detection. In retrospect, using the set of defects discovered during a software product's lifetime, the quality of a test suite could be evaluated by measuring its ability to detect those faults (faults never revealed in use might reasonably have little impact on testing decisions). Of course, this is not a practical method for making decisions during development and testing. Software engineers therefore rely on methods that predict fault detection capability based only on the suite itself and the current version of the software under test (SUT). The most popular such method is the use of code coverage criteria [1]. Code coverage describes structural aspects of the executions of an SUT performed by a test suite. For example, statement coverage indicates which statements in a program's source code were executed, branch coverage indicates which branches were taken, and path coverage describes (typically in a slightly more complex way, to account for loops) the paths explored in a program's control flow graph.

In software testing research, the gold standard for suite evaluation is generally considered to be actual faults detected, but this is, again, in practice difficult to apply even in a research setting [16]. The second most informative measure of suite quality is usually believed to be *mutation testing* [7, 2], which measures the ability of a test suite to detect small changes to the source code. Mutation testing subsumes many other code coverage criteria, and has been shown to predict actual fault detection better than other criteria in some settings, but never shown to be worse than traditional code coverage measures.

Unfortunately, mutation testing is both difficult to apply and computationally expensive, which has led to the search for "next-best" criteria for predicting suite quality by researchers [16, 19]. This effort is highly relevant to real software developers, who almost never apply mutation testing due to its complexity, expense, and the lack of tool support in many languages. From the point of view of actual software developers and test engineers, rather than researchers,

High coverage may not mean much, but **low coverage** means a lot!

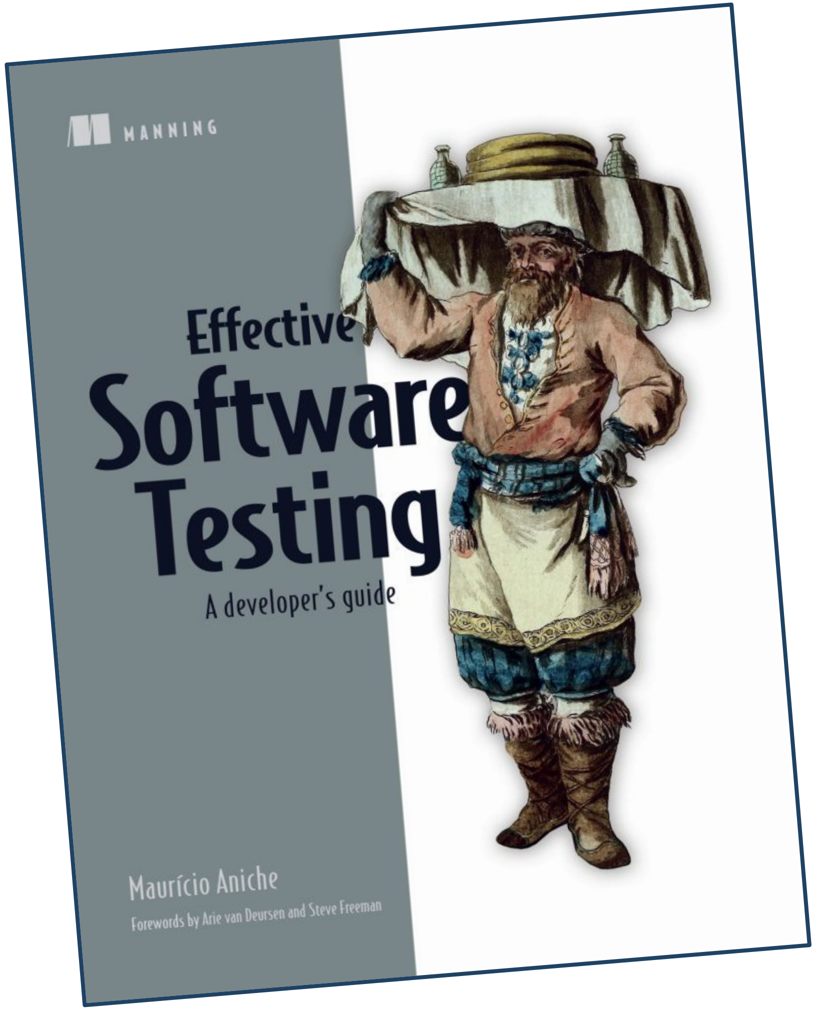
Maurício Aniche

@mauricioaniche

www.effective-software-testing.com

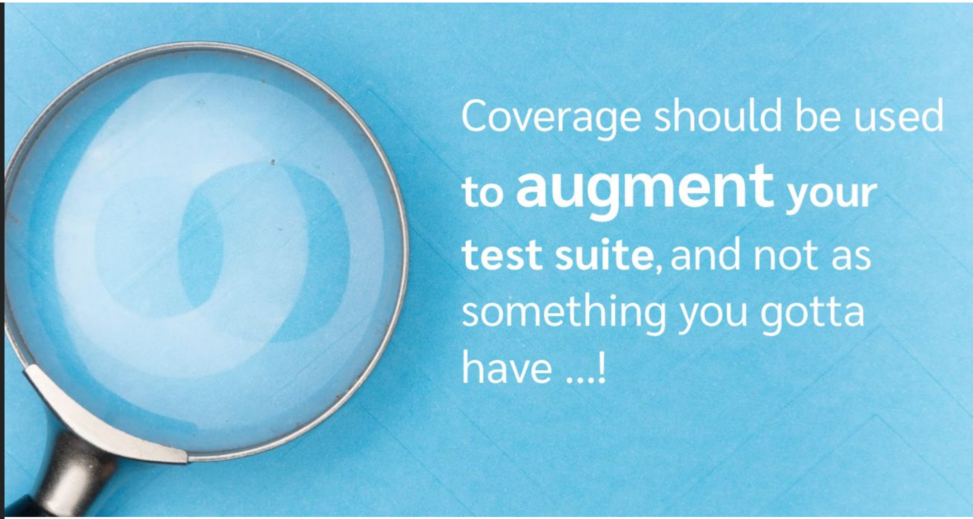
Use “**au35ani**” discount for 35% off!

Subscribe to my newsletter!


MANNING
Effective Software Testing
A developer's guide
Maurício Aniche
Forewords by Arie van Deursen and Steve Freeman

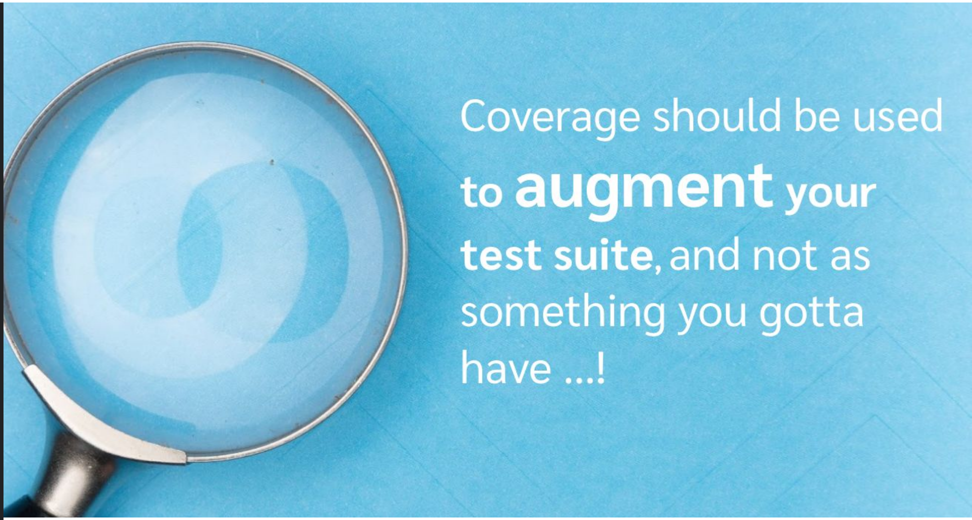
If you hate code coverage, that's because you **are not** using it properly!

If you hate code coverage, that's because you **are not** using it properly!



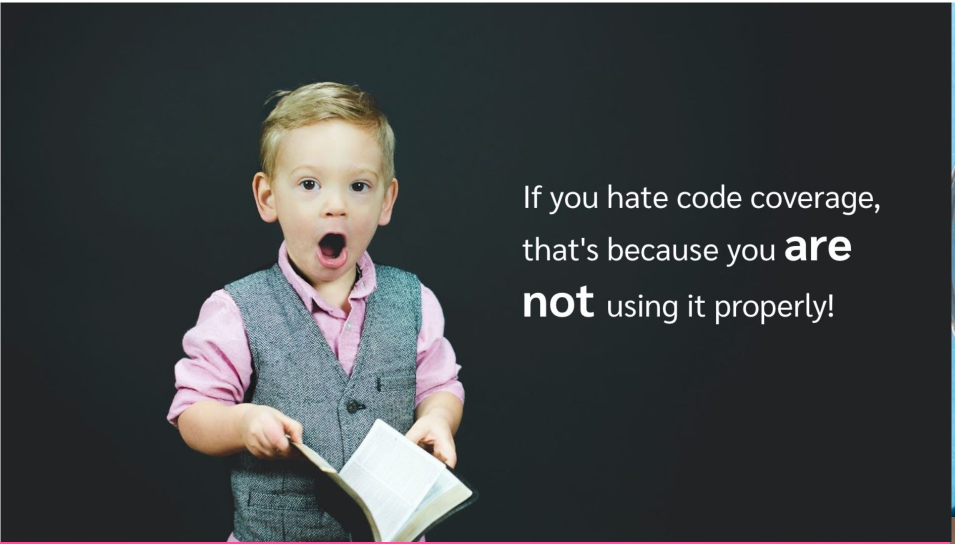
Coverage should be used to **augment your test suite**, and not as something you gotta have ...!

If you hate code coverage, that's because you **are not** using it properly!

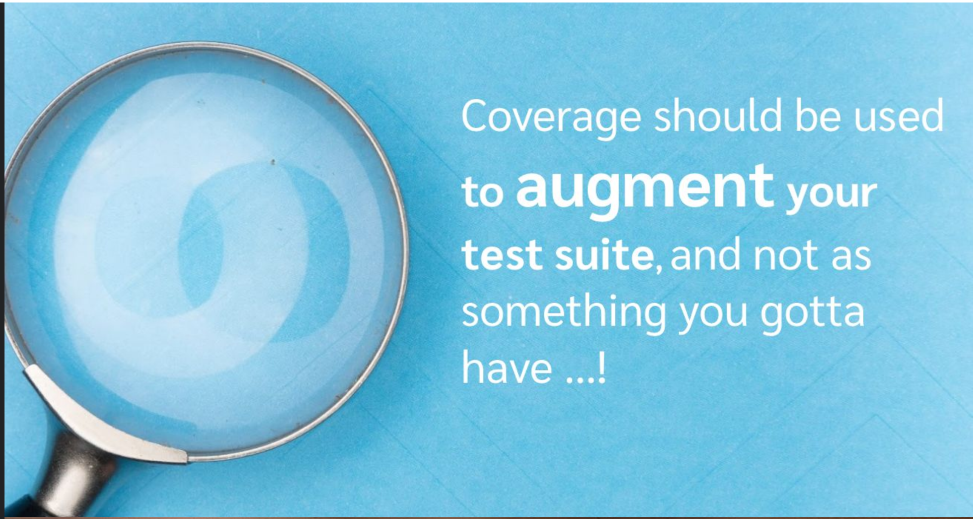


Coverage should be used to **augment your test suite**, and not as something you gotta have ...!

Code coverage **informs the decision**, but it shouldn't take the decision for you!



If you hate code coverage, that's because you **are not** using it properly!



Coverage should be used to **augment** your **test suite**, and not as something you gotta have ...!

Code coverage **informs the decision**, but it shouldn't take the decision for you!

High coverage may not mean much, but **low coverage** means a lot!

License