



S. SCALABRINO
M. GUERRA
R. OLIVETO



G. GRANO
H. GALL



D. Di NUCCI



A. DE LUCIA



OCELOT

A SEARCH-BASED TEST-DATA GENERATION TOOL FOR C



Testing
is expensive

Automatic Test Generation

UNIVERSITY OF MICHIGAN

Search-Based Test Generation

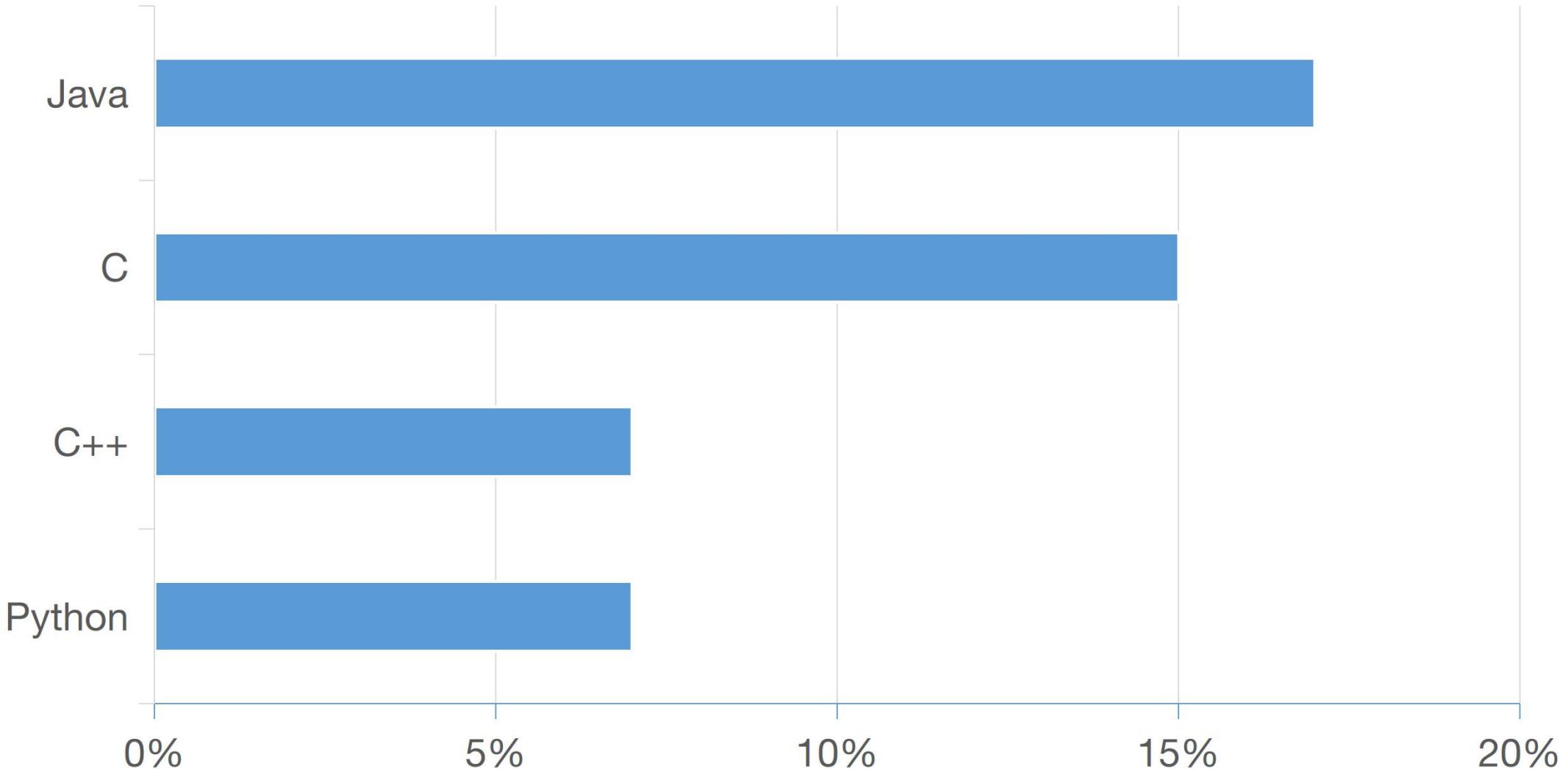




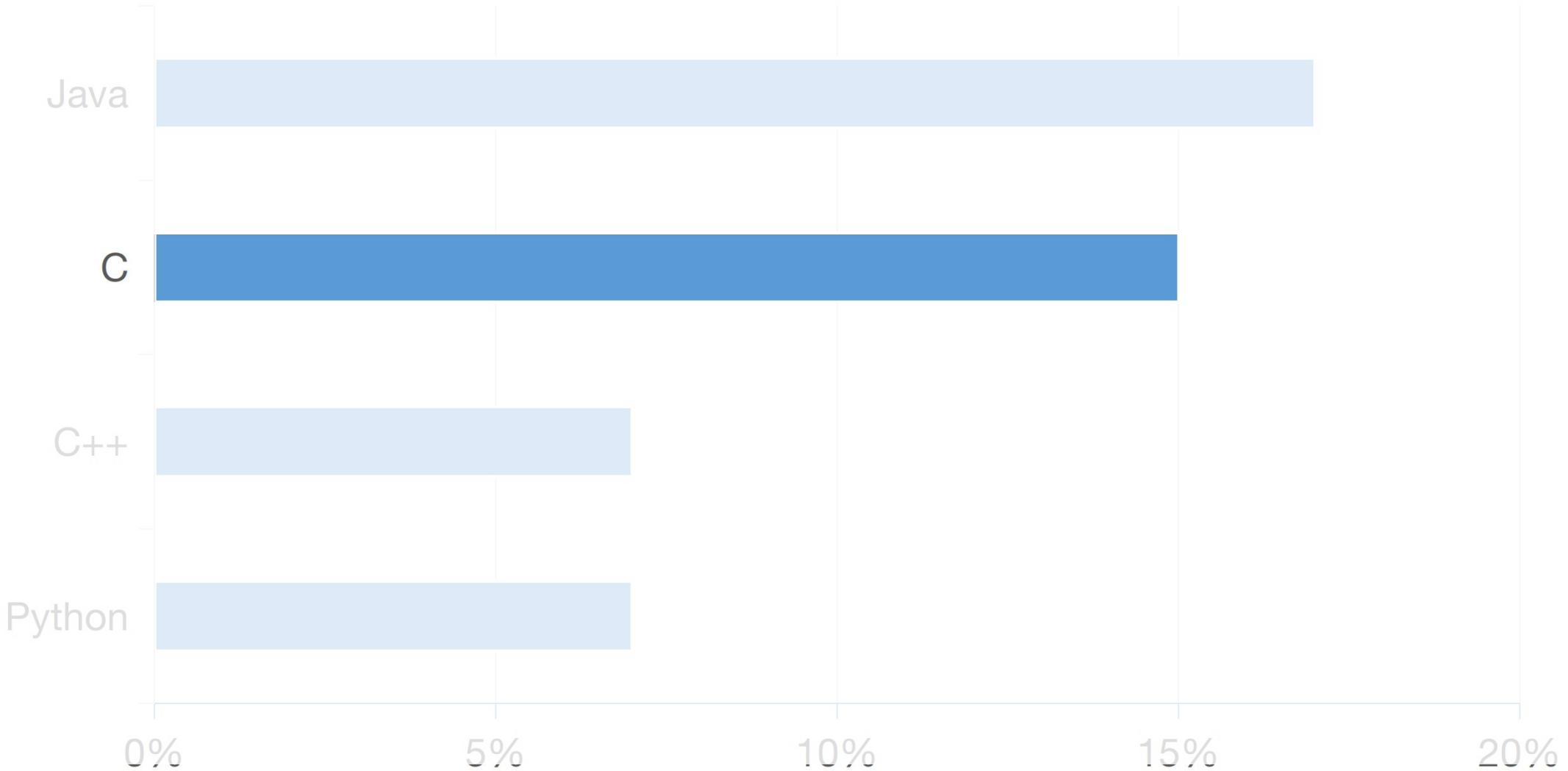
Randoop

Automatic unit test generation for Java

TIOBE Ranking



TIOBE Ranking



TESTGEN

QUEST

GADGET

IGUANA

Austin

CAVM

TESTGEN

QUEST

GADGET

IGUANA

Austin

CAVM



OCELOT



SOURCE FUNCTION



TEST SUITE

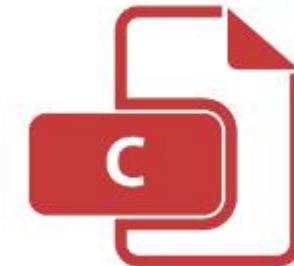
OCELOT



SOURCE FUNCTION



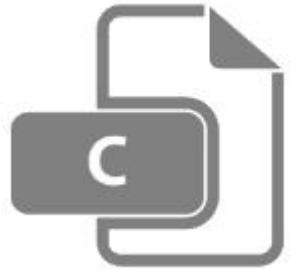
OCELOT



TEST SUITE

BUILD + RUN + WRITE

BUILD



SOURCE FUNCTION

BUILD



SOURCE FUNCTION



INSTRUMENTED CODE

BUILD



SOURCE FUNCTION

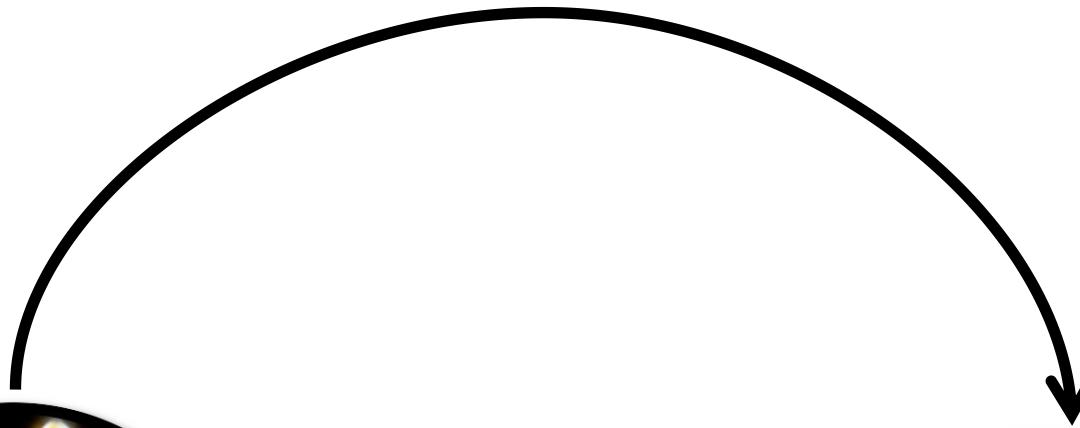


INSTRUMENTED CODE



COMPILED BINARY

RUN



COMPILED BINARY

RUN



COMPILED BINARY



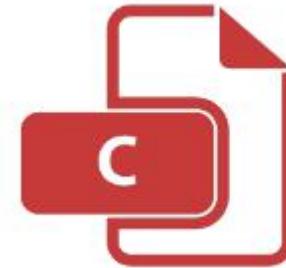
WRITE



WRITE



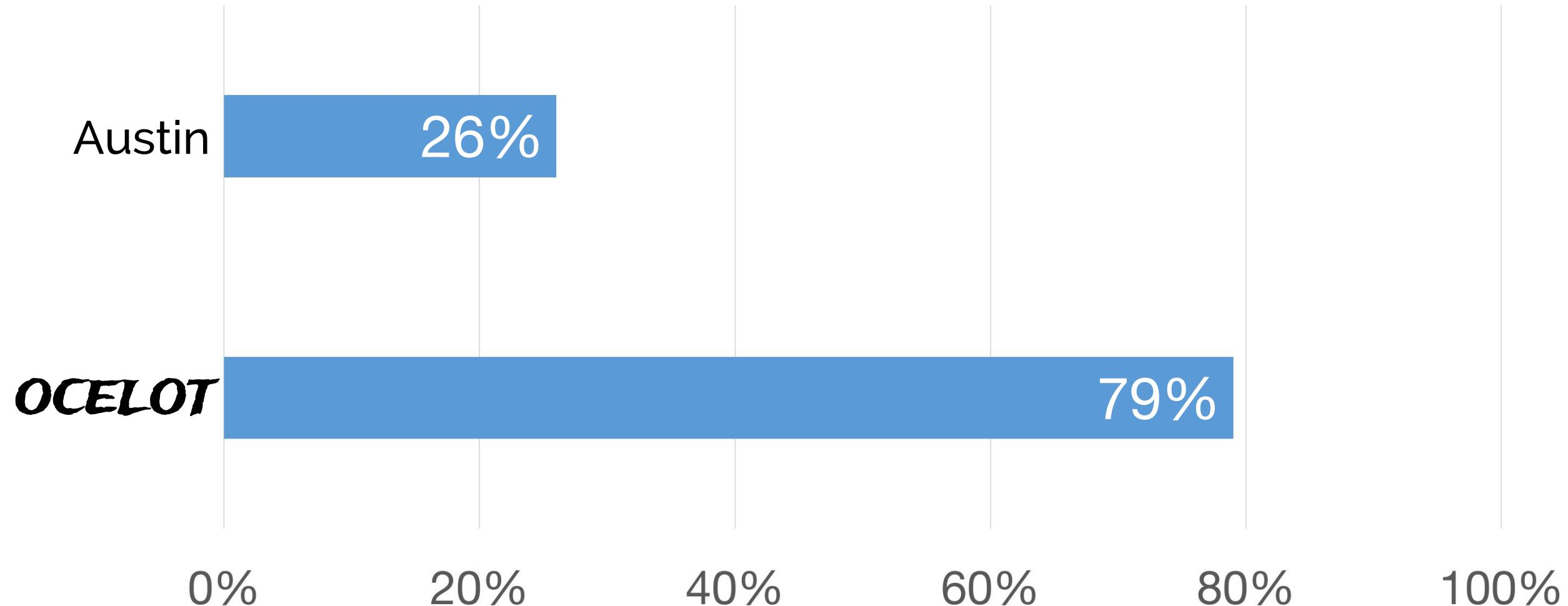
```
#include <check.h>
```

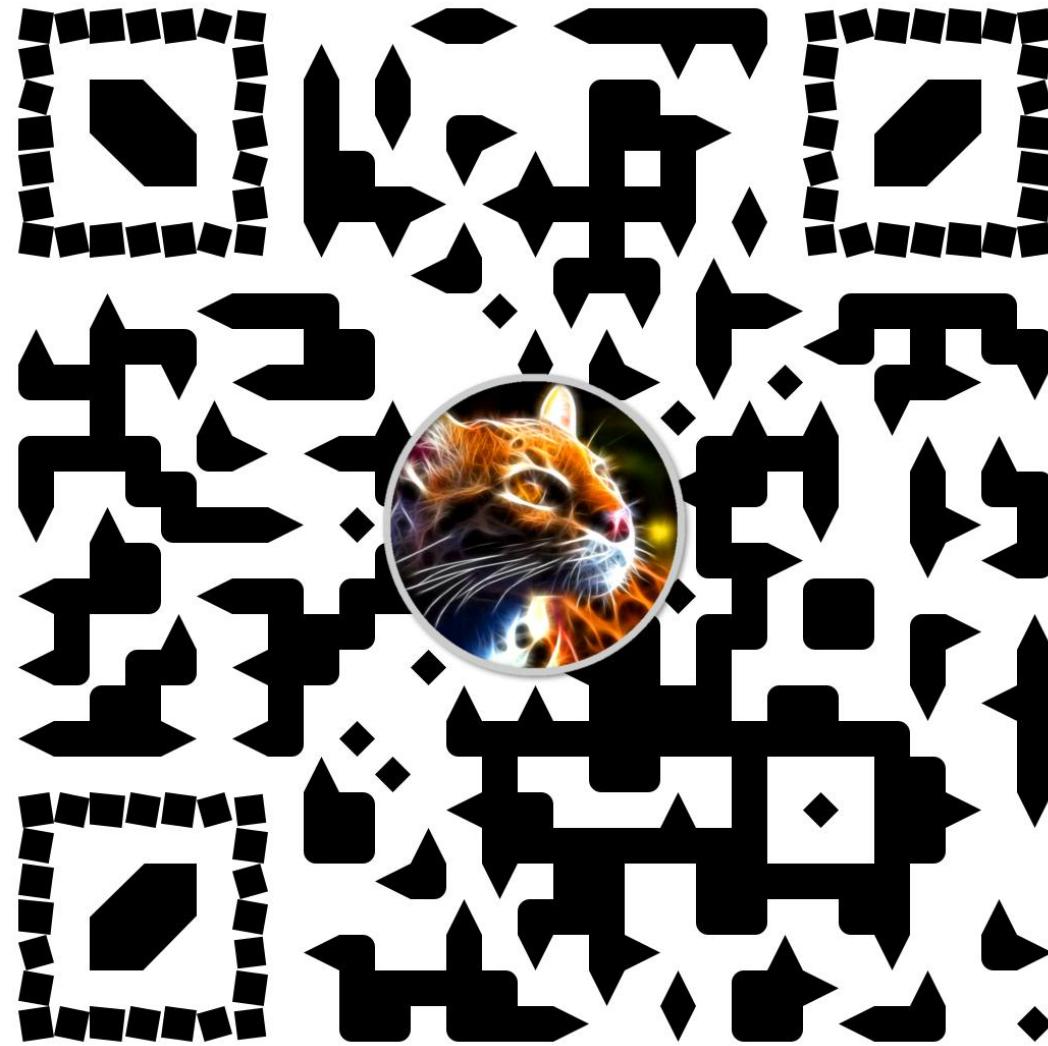


TEST SUITE

OCELOT vs Austin

Branch Coverage





<https://ocelot.science>



HANDS-ON DEMO
FRIDAY, 10:30 - 12:00
ROOM JOFFRE CD



OCELOT

A SEARCH-BASED TEST-DATA GENERATION TOOL FOR C



S. SCALABRINO

M. GUERRA

R. OLIVETO



G. GRANO

H. GALL



D. Di NUCCI



A. DE LUCIA

Come see the demo (0.5 min)

Abstract
We present a new tool, named DART, for automatically testing software that combines three main techniques: (1) automated extraction of test cases from static source code using static source-code parsing; (2) automatic generation of a test driver that runs the program under test and captures its output in the most general environment the program can operate in; and (3) dynamic analysis of how the program behaves under random test inputs. The DART tool can automatically generate test cases, including the execution along alternative program paths. Together, these three features allow DART to automatically generate tests for DART itself for short. The main strength of DART is thus that testing can be performed *completely automatically* on any program that complies with the constraints of the tool. In addition to automated testing, DART detects standard errors such as program crashes, assert violations, and memory leaks. The results of our experiments show that unit test several examples of C programs are very encouraging.

Categories and Subject Descriptors D.2.4 [Software Engineering]: Software Verification; D.2.5 [Software Engineering]: Testing and Debugging; F.3.1 [Mathematical Foundations]: Specification and Verifying and Reasoning about Programs

General Terms Verification, Algorithms, Reliability

Keywords Software Testing, Random Testing, Automated Test Generation, Interfaces, Program Verification

1. Introduction
Today, testing is the primary way to check the correctness of software. Testing is a major cost factor in the software industry, as testing usually accounts for about 50% of the cost of software development. According to the Software Quality Institute, currently the US economy alone spends \$60 billion every year, and that implements in software testing infrastructure may save one-third of that amount.

Among the various kinds of testing usually performed during software development, unit testing is the most common for individual components of a software system. In principle, unit testing plays an important role in ensuring overall software quality since it checks the correctness of individual components in each block, all corner cases, and provide 100% code coverage. Yet, in practice,

Search
Because the first applied I was able to start testing an increasing number that prove structural

to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. The digital version of this article is available at [http://www.acm.org/journals/pjtc.html](#). To copy otherwise, to post on servers or to redistribute to lists, requires prior permission and/or a fee. Request permission to photocopy or

IGUANA: Input Generation Using Automated Novel Algorithms. A Plug and Play Research Tool

Phil McNamee
University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello Street,
Sheffield, S1 4DP, UK
p.mcnamee@cs.shef.ac.uk

Abstract
IGUANA is a tool for automatically generating software test data using search-based approaches. Search-based approaches explore the input domain of a program for test data and are guided by a fitness function. The fitness function evaluates input data and measures how suitable it is for the purpose of the test. This could be the detection of a bug in a program, or the validation of an assertion statement.

The IGUANA tool is designed so that researchers can easily compare and contrast different search-based approaches, including local search, genetic algorithms, fitness functions (e.g., for obtaining branch coverage of a program) and program analysis techniques for test data generation.

1. Introduction
In contrast to methods like symbolic execution, search-based techniques take a heuristic approach to the test data generation problem. Symbolic execution needs to extract a series of constraints from the program that describe the execution of a particular path. These constraints are solved using linear programming techniques. A search-based approach to this problem instead involves designing a *fitness function* which essentially gives a measure of how close the test data were to meeting the requirements of the test. The search is the process of exploring the parameter space implemented in order to feedback information to the search algorithm for fitness function computation. In this way, some of the problems associated with symbolic execution, e.g. the handling of loops and conditionals, are avoided.

There has recently been an explosion of work in the area search-based testing, and in particular the generation of test data. Search-based approaches have been shown to be an effective approach for fault localization [15], model checking [20, 14, 21], automated [7, 8, 3, 22, 5, 13, 34, 11, 19], and cover-based [9, 18] testing. McNamee et al. [30] proposed an open source implementation in Python in the area, mainly concentrating on structural test data generation. To date, several search algorithms have been employed, including random search, local search (including hill climbing, tabu search, simulated annealing), local search with mutation operators, genetic algorithms, ant colony optimisation strategies and genetic programming, ant colony optimisation and particle swarm optimisation; search-based approaches have also been used for fault localization, model checking, functional testing, and non-functional properties such as worse-case execution time, stress-based testing, and so on. Techniques such as testability transformations [4] have been proposed to circumvent certain search-based testing problems, for example plateau in the fitness landscape.

Until now, there has not been a general framework for comparing different search methods and techniques on the test data generation problem. It is has therefore been difficult to judge results presented in the literature. Since different implementations of algorithms have been used, with

AUSTIN
Kiran L.
*CREST, Univ.
Berkeley & M.
A R T I C L E
Article Jour.
Volume 36
Number 1
Received 10
SBE
MSIT

Abstract
We present a new tool, named DART, for automatically testing software that combines three main techniques: (1) automated extraction of test cases from static source code using static source-code parsing; (2) automatic generation of a test driver that runs the program under test and captures its output in the most general environment the program can operate in; and (3) dynamic analysis of how the program behaves under random test inputs. The DART tool can automatically generate test cases, including the execution along alternative program paths. Together, these three features allow DART to automatically generate tests for DART itself for short. The main strength of DART is thus that testing can be performed *completely automatically* on any program that complies with the constraints of the tool. In addition to automated testing, DART detects standard errors such as program crashes, assert violations, and memory leaks. The results of our experiments show that unit test several examples of C programs are very encouraging.

Categories and Subject Descriptors D.2.4 [Software Engineering]: Software Verification; D.2.5 [Software Engineering]: Testing and Debugging; F.3.1 [Mathematical Foundations]: Specification and Verifying and Reasoning about Programs

General Terms Verification, Algorithms, Reliability

Keywords Software Testing, Random Testing, Automated Test Generation, Interfaces, Program Verification

1. Introduction
Today, testing is the primary way to check the correctness of software. Testing is a major cost factor in the software industry, as testing usually accounts for about 50% of the cost of software development. According to the Software Quality Institute, currently the US economy alone spends \$60 billion every year, and that implements in software testing infrastructure may save one-third of that amount.

Among the various kinds of testing usually performed during software development, unit testing is the most common for individual components of a software system. In principle, unit testing plays an important role in ensuring overall software quality since it checks the correctness of individual components in each block, all corner cases, and provide 100% code coverage. Yet, in practice,

Search
Because the first applied I was able to start testing an increasing number that prove structural

to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage. The digital version of this article is available at [http://www.acm.org/journals/pjtc.html](#). To copy otherwise, to post on servers or to redistribute to lists, requires prior permission and/or a fee. Request permission to photocopy or

IGUANA: Input Generation Using Automated Novel Algorithms. A Plug and Play Research Tool

Phil McNamee
University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello Street,
Sheffield, S1 4DP, UK
p.mcnamee@cs.shef.ac.uk

Abstract
IGUANA is a tool for automatically generating software test data using search-based approaches. Search-based approaches explore the input domain of a program for test data and are guided by a fitness function. The fitness function evaluates input data and measures how suitable it is for the purpose of the test. This could be the detection of a bug in a program, or the validation of an assertion statement.

The IGUANA tool is designed so that researchers can easily compare and contrast different search-based approaches, including local search, genetic algorithms, fitness functions (e.g., for obtaining branch coverage of a program) and program analysis techniques for test data generation.

1. Introduction
In contrast to methods like symbolic execution, search-based techniques take a heuristic approach to the test data generation problem. Symbolic execution needs to extract a series of constraints from the program that describe the execution of a particular path. These constraints are solved using linear programming techniques. A search-based approach to this problem instead involves designing a *fitness function* which essentially gives a measure of how close the test data were to meeting the requirements of the test. The search is the process of exploring the parameter space implemented in order to feedback information to the search algorithm for fitness function computation. In this way, some of the problems associated with symbolic execution, e.g. the handling of loops and conditionals, are avoided.

There has recently been an explosion of work in the area search-based testing, and in particular the generation of test data. Search-based approaches have been shown to be an effective approach for fault localization [15], model checking [20, 14, 21], automated [7, 8, 3, 22, 5, 13, 34, 11, 19], and cover-based [9, 18] testing. McNamee et al. [30] proposed an open source implementation in Python in the area, mainly concentrating on structural test data generation. To date, several search algorithms have been employed, including random search, local search (including hill climbing, tabu search, simulated annealing), local search with mutation operators, genetic algorithms, ant colony optimisation strategies and genetic programming, ant colony optimisation and particle swarm optimisation; search-based approaches have also been used for fault localization, model checking, functional testing, and non-functional properties such as worse-case execution time, stress-based testing, and so on. Techniques such as testability transformations [4] have been proposed to circumvent certain search-based testing problems, for example plateau in the fitness landscape.

Until now, there has not been a general framework for comparing different search methods and techniques on the test data generation problem. It is has therefore been difficult to judge results presented in the literature. Since different implementations of algorithms have been used, with

1