



INTRODUCTION TO SOFTWARE TESTING FOR THE SCIENTIFIC COMMUNITY

Myra Cohen, BSSw Fellow

<https://www.cs.iastate.edu/~mcohen>

mcohen@iastate.edu



IOWA STATE UNIVERSITY
OF SCIENCE AND TECHNOLOGY

Laboratory for Variability-Aware Assurance and
Testing of Organic Programs



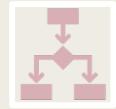
Module I: Overview



Motivation



What to test



Types of
Testing



Models



Coverage



Oracles

Why Test Software?

```
Error: NCBI C++ Exception:
```

```
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/objistrasnbcpp"
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/member.cpp", l-
```

Why Test Software?

```
Error: NCBI C++ Exception:
```

```
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/objistrasn
```

```
ial/objistrasn
```

```
.cpp", line 499: Error: (CSerialException::eOverflow) byte 132: overf
```

```
ial/member.cpp", line 767: Error: (CSerialException::eOverflow) ncbi::CMemberInfoFur
```

Why Test Software?

NCBI blastp bug - changing max_target_seqs returns incorrect top hits

[2015-11-30-blastp-bug.md](#)

Raw

NCBI blastp seems to have a bug where it reports different top hits when -max_target_seqs is changed. This is a serious problem because the first 20 hits (for example) should be the same whether -max_target_seqs 100 or -max_target_seqs 500 is used.

The bug is reproducible on the command line when searching NCBI's nr blast database (dated 25-Nov-2015) using NCBI 2.2.28+, 2.2.30+ and 2.2.31+.

Why Test Software?

NCBI blastp bug - changing max_target_seqs returns incorrect top hits

2015-11-30-blastp-bug.md Raw

NCBI blastp problem I
500 is us
The bug
2.2.28+,

Bioinformatics, 35(9), 2019, 1613–1614
doi: 10.1093/bioinformatics/bty833
Advance Access Publication Date: 24 September 2018
Letter to the Editor

OXFORD

serious
et_seqs
ig NCBI

Sequence analysis

Misunderstood parameter of NCBI BLAST impacts the correctness of bioinformatics workflows

Nidhi Shah¹, Michael G. Nute², Tandy Warnow  ³ and Mihai Pop  ^{1,*}

¹Department of Computer Science, University of Maryland College Park, MD 20742, USA, ²Department of Statistics and ³Department of Computer Science, University of Illinois Urbana-Champaign, Champaign, IL 61820, USA

*To whom correspondence should be addressed.
Associate Editor: John Hancock
Contact: mpop@umd.edu

Received and revised on August 13, 2018; editorial decision on September 19, 2018; accepted on September 21, 2018

Why Test Software?

Bioinformatics, 35(15), 2019, 2699–2700

doi: 10.1093/bioinformatics/bty1026

Advance Access Publication Date: 24 December 2018

Letter to the Editor



Sequence analysis

Reply to the paper: Misunderstood parameters of NCBI BLAST impacts the correctness of bioinformatics workflows

Thomas L. Madden*, Ben Busby and Jian Ye

National Center for Biotechnology Information, National Library of Medicine, National Institutes of Health,
Bethesda, MD 20894, USA

*To whom correspondence should be addressed.

Associate Editor: John Hancock

Contact: madden@ncbi.nlm.nih.gov

Received and revised on November 30, 2018; editorial decision on December 10, 2018; accepted on December 19, 2018

Why Test Software?

Dear Editor,

A recent letter by [Shah et al. \(2018\)](#) addressed the use of a command-line parameter in BLAST ([Altschul et al., 1997](#); [Camacho et al., 2009](#)). BLAST is a very popular tool, so it is not surprising that this topic has provoked a great deal of interest. The authors have, however, conflated three different issues. One is a bug that will be fixed in the BLAST+ 2.8.1 release due out in December 2018, another is simply how BLAST works and the third might be viewed as a [shortcoming of our implementation of composition-based statistics \(CBS\)](#). Here, we address these issues and describe some new documentation about the BLAST process.

[Shah et al. \(2018\)](#) did not provide their own example in the letter, but later provided one at https://github.com/shahnidhi/BLAST_maxtargetseq_analysis. At the NCBI, we examined the new example and it became clear that the demonstrated behavior was a bug, resulting from an overly aggressive optimization, introduced in 2012 for BLASTN and MegaBLAST (DNA–DNA alignments). This bug has been fixed in the BLAST+ 2.8.1 release, due out in December 2018. The aberrant behavior seems to occur only in alignments with an extremely large number of gaps, which is the case in the example provided by Shah and collaborators.

Why Test Software?

contributed articles



DOI:10.1145/3382037

An approach to reproducibility problems related to porting software across machines and compilers.

BY DONG H. AHN, ALLISON H. BAKER, MICHAEL BENTLEY, IAN BRIGGS, GANESH GOPALAKRISHNAN, DORIT M. HAMMERLING, IGNACIO LAGUNA, GREGORY L. LEE, DANIEL J. MILROY, AND MARIANA VERTENSTEIN

Keeping Science on Keel When Software Moves

the machine instructions that actually get executed. Unfortunately, such changes do affect the computed results to a significant (and often worrisome) extent. In a majority of cases, there are not easily definable *a priori* answers one can check against. A programmer ends up comparing the new answer against a trusted baseline previously established or checks for indirect confirmations such as whether physical properties such as energy are conserved. However, such non-systematic efforts might miss underlying issues, and the code may keep misbehaving until these are fixed.

In this article, we present real-world evidence to show that ignoring numerical result changes can lead to misleading scientific conclusions. We present techniques and tools that can help computational scientists understand and analyze compiler effects on their scientific code. These techniques are applicable across a wide range of examples to narrow down the root-causes to single files, functions within files, and even computational expressions that affect specific variables. The developer may then rewrite the code selectively and/or suppress the application of certain optimizations to regain more predictable behavior.

Going forward, the frequency of required ports of computational software will increase, given that performance gains can no longer be obtained by mere-

Why Test Software?

contributed articles



An approach to reproducibility problems related to porting software across machines and compilers.

BY DONG H. AHN, ALLISON H. BAKER, MICHAEL BENTLEY, IAN BRIGGS, GANESH GOPALAKRISHNAN, DORIT M. HAMMERLING, IGNACIO LAGUNA, GREGORY L. LEE, DANIEL J. MILROY, AND MARIANA VERTENSTEIN

DOI:10.1145/3382037

the machine instructions that actually get executed. Unfortunately, such changes do affect the computed results to a significant (and often worrisome) extent. In a majority of cases, there are not easily definable *a priori* answers one can check against. A programmer ends up comparing the new answer against a trusted baseline previously established or checks for indirect confirmations such as whether physical properties such as energy are conserved. However, such non-systematic efforts might miss underlying issues, and the code may keep misbehaving until these are fixed.

In this article, we present real-world evidence to show that ignoring numerical result changes can lead to misleading scientific conclusions. We present techniques and tools that can help computational scientists understand and analyze compiler effects on their scientific code. These techniques are applicable across a wide range of examples to narrow down the root-causes to single files, functions within files, and even computational expressions that affect specific variables. The developer may then rewrite the code selectively and/or suppress the application of certain optimizations to regain more predictable behavior.

Going forward, the frequency of required ports of computational software will increase, given that performance gains can no longer be obtained by mere-

Keeping Science on Keel When Software Moves

In this article, we present real-world evidence to show that ignoring numerical result changes can lead to misleading scientific conclusions. We present tech-

Why Test Software?

NCBI blastp bug - changing max_target_seqs returns incorrect top hits

[2015-11-30-blastp-bug.md](#) Raw

NCBI blastp seems to have a bug where it reports different top hits when -max_target_seqs is changed. This is a serious problem because the first 20 hits (for example) should be the same whether -max_target_seqs 100 or -max_target_seqs 500 is used.

The bug is reproducible on the command line when searching NCBI's nr blast database (dated 25-Nov-2015) using NCBI 2.2.28+, 2.2.30+ and 2.2.31+.

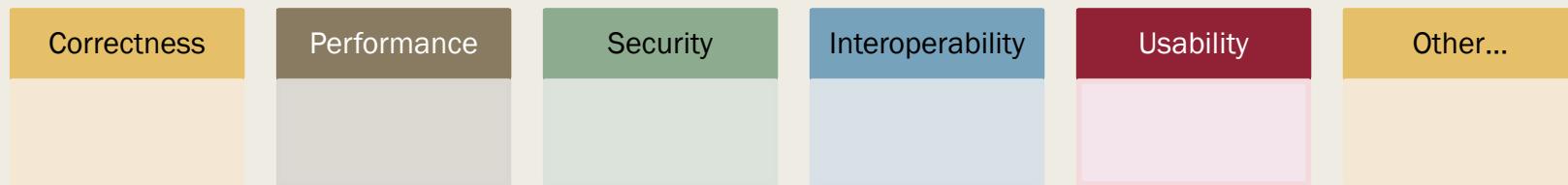
[Shah et al. \(2018\)](#) did not provide their own example in the letter, but later provided one at https://github.com/shahnidhi/BLAST_maxtargetseq_analysis. At the NCBI, we examined the new example and it became clear that the demonstrated behavior was a bug, resulting from an overly aggressive optimization, introduced in 2012 for BLASTN and MegaBLAST (DNA–DNA alignments). This bug has been fixed in the BLAST+ 2.8.1 release, due out in December 2018. The aberrant behavior seems to occur only in alignments with an extremely large number of gaps, which is the case in the example provided by Shah and collaborators.

Why Test Software?

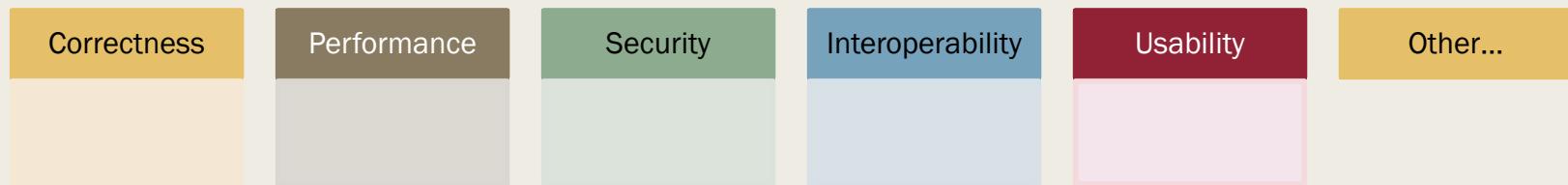


Merali, Zeeya. "Computational Science: ...Error." *Nature* 467, no. 7317 (Oct, 2010): 775–77

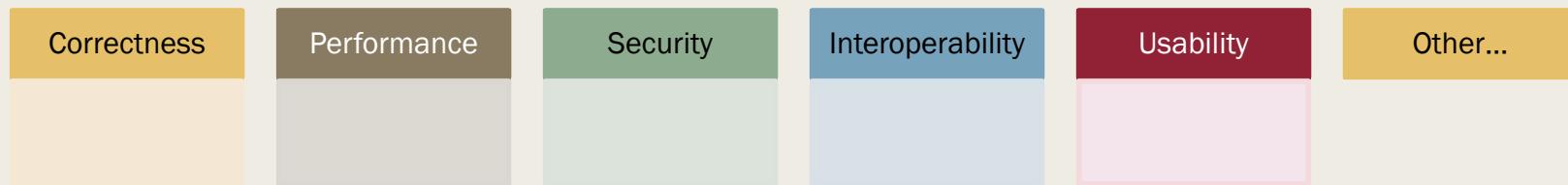
What Should We Test?



What Should We Test?



What Should We Test?

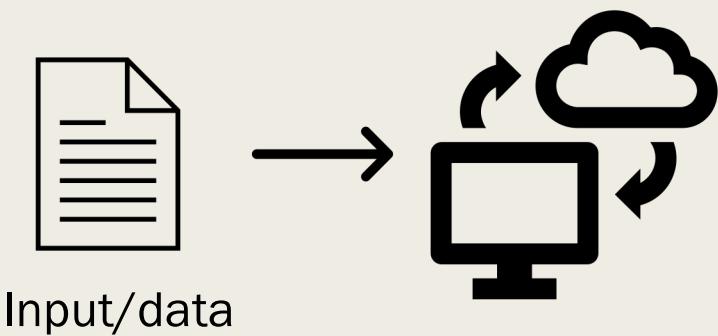


What is Testing

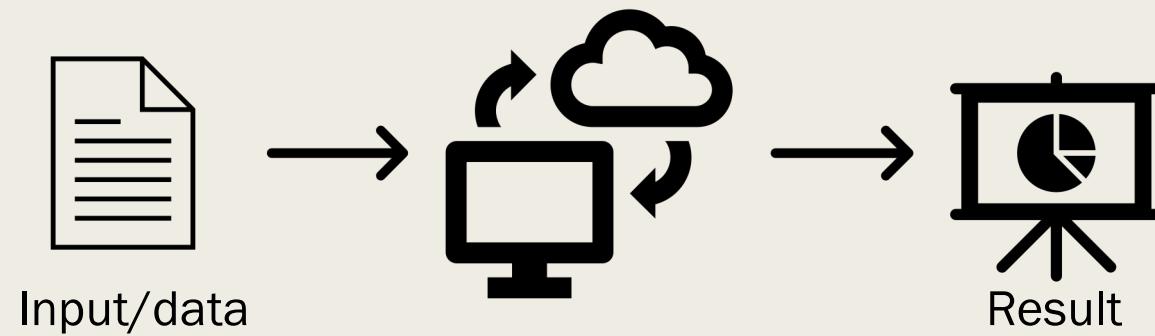


Input/data

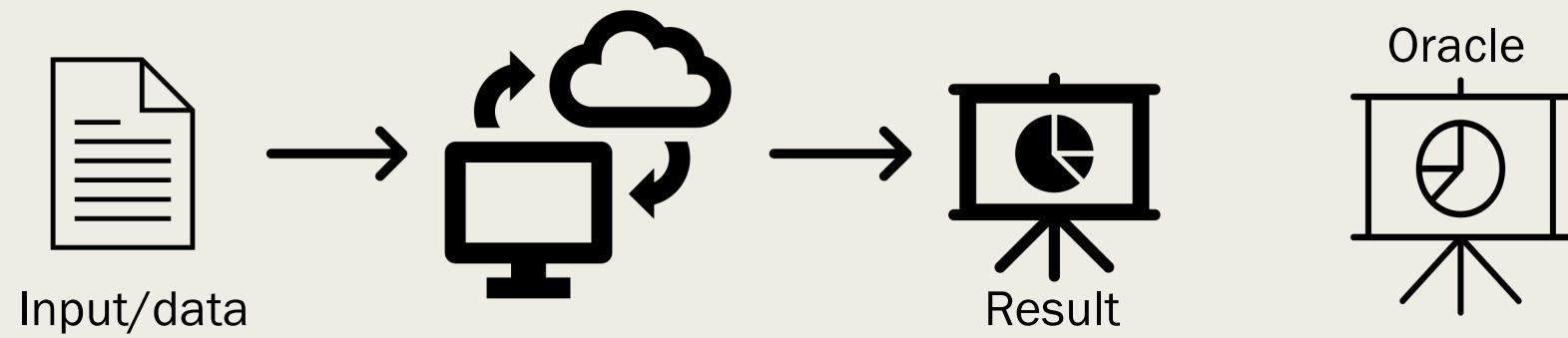
What is Testing



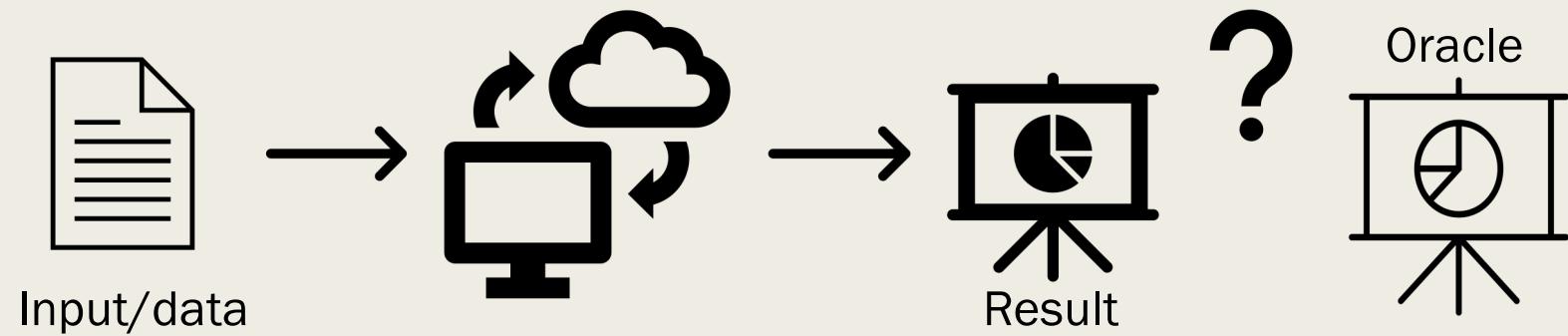
What is Testing



What is Testing



What is Testing



Challenges

- To detect a program FAILURE we need to:
 - Reach a FAULT in the code
 - Infect the code (change to incorrect state) - ERROR
 - Propagate the error out of program
 - Reveal (detect) the error – (ORACLE)

RIPR model Ammann, Offutt (*Introduction to Software Testing, 2016*)

Tests Can Miss Faults

```
def classify_triangle(a, b, c):
    # Sort the sides so that a <= b <= c
    if a > b:
        tmp = a
        a = tmp  #fault should be a=b
        b = tmp

    if a > c:
        tmp = a
        a = c
        c = tmp

    if b > c:
        tmp = b
        b = c
        c = tmp

    if a + b <= c:
        return TriangleType.INVALID
    elif a == b and b == c:
        return TriangleType.EQUILATERAL
    elif a == b or b == c:
        return TriangleType.ISOSCELES
    else:
        return TriangleType.SCALENE
```

1. Test Case 3, 4, 5
 - doesn't reach fault
2. Test Case 5, 1, 1
 - reaches fault and infects
 - reveals (returns isosceles)
3. Test Case 2, 1, -1
 - reaches fault and infects
 - Doesn't propagate (-1,2,2) returns INVALID

Software vs. Data



Scientific software is often data (or model driven)



Both software and data can lead to faults



We start by **focusing on software**

But I Have Unit Tests...



These are an essential part of testing



Focus is on individual modules



Can be re-used each time system changes (regression testing)



Can be packaged with software when released

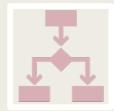
Module I: Overview



Motivation



What to test



Types of
Testing



Models



Coverage



Oracles

Types of Testing



Unit Testing



Integration
Testing



System
Testing



Configuration
Testing



User Interface
Testing



Regression
Testing

Models



Provide an **abstraction** of the software we are testing



Can be **for different dimensions** of the software (specifications, interface, code)



Allow us to reason about **how much** we have tested



The foundation for **automated test generation**

Example Models

Graphs

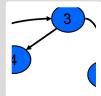
Tabular

Relational

Grammar based

Logic based

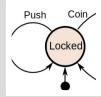
Graph Models



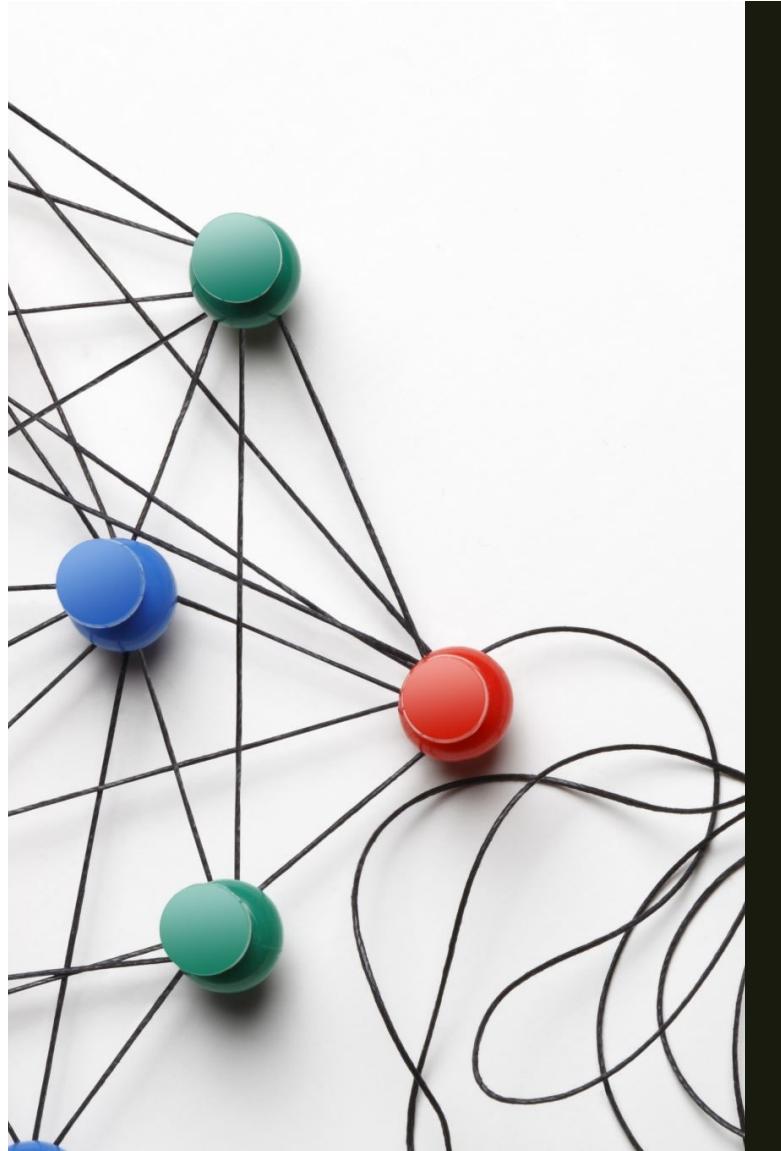
Program control flow graph



User interface



Program state machine



Types of Graph Coverage

- All **nodes**
- All **edges** (pairs of nodes)
- All **length N** paths
- **M random** length N paths

Program Code Coverage

Cog coverage: 38.75%

coverage.py v7.2.7, created at 2023-05-29 15:26 -0400

Module	statements	missing	excluded	branches	partial	coverage
cogapp/__init__.py	1	0	0	0	0	100.00%
cogapp/__main__.py	3	3	0	0	0	0.00%
cogapp/cogapp.py	500	224	1	210	30	49.01%
cogapp/makefiles.py	22	18	0	14	0	11.11%
cogapp/test_cogapp.py	845	591	2	24	1	29.57%
cogapp/test_makefiles.py	70	53	0	6	0	22.37%
cogapp/test_whiteweutils.py	68	50	0	0	0	26.47%
cogapp/whiteutils.py	43	5	0	34	4	88.31%
Total	1552	944	3	288	35	38.75%

coverage.py v7.2.7, created at 2023-05-29 15:26 -0400

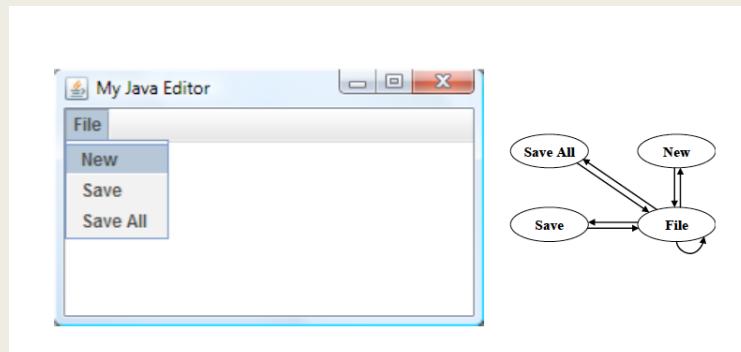
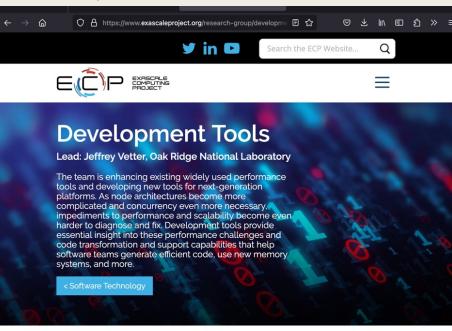
Triangle.java

```
1. public class Triangle {
2.
3.     public enum TriangleType {
4.         INVALID, SCALENE, EQUILATERAL, ISOSCELES
5.     }
6.
7.     public static TriangleType classifyTriangle(int a, int b, int c) {
8.         if (a > b) {
9.             int tmp = a;
10.            a = b;
11.            b = tmp;
12.        }
13.        if (a > c) {
14.            int tmp = c; // original: int tmp = a;
15.            a = c;
16.            c = tmp;
17.        }
18.        if (b > c) {
19.            int tmp = b;
20.            b = c;
21.            c = tmp;
22.        }
23.        if (a + b <= c) {
24.            return TriangleType.INVALID;
25.        } else if (a == b && b == c) {
26.            return TriangleType.EQUILATERAL;
27.        } else if (a == b || b == c) {
28.            return TriangleType.ISOSCELES;
29.        } else {
30.            return TriangleType.SCALENE;
31.        }
32.    }
33.
34.
35.
36.
37. }
```

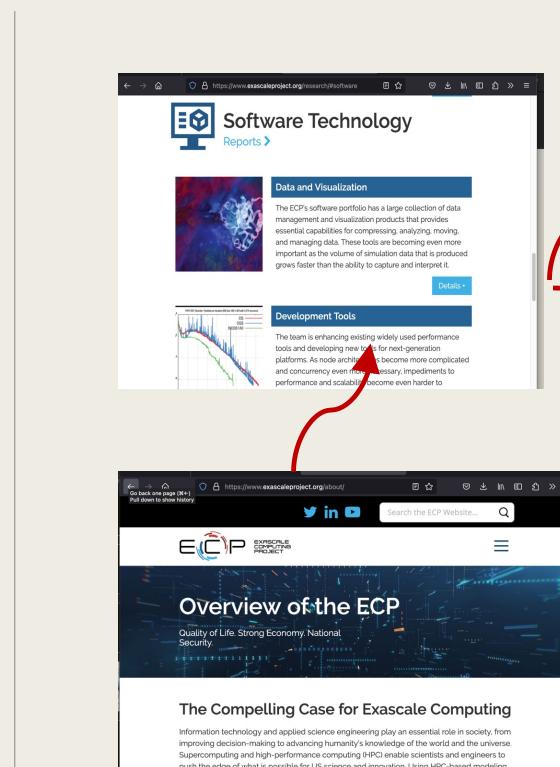
Example tools: jacoco, coverage.py, gcov

Interface (graph) Coverage

Web



GUI



Other Coverage

Specification coverage

- Cover the system requirements

Interaction coverage

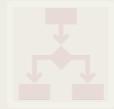
Module I: Overview



Motivation



What to test



Types of
Testing



Models

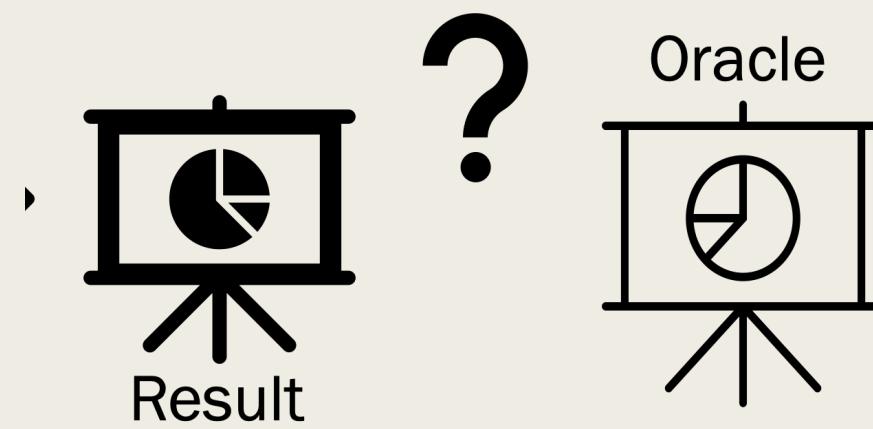


Coverage



Oracles

What is the Correct Answer?



Trivial Oracles

Program crashes

Core dump

Segmentation error

Overflow

Program hangs

Trivial Oracles

- Good when we don't have a known result
- Weakest oracle since it only shows that the program fails/not that the result is incorrect
- Exact oracles are easy to compute in some programs

```
def classify_triangle(a, b, c):  
    # Sort the sides so that a <= b <= c  
    if a > b:  
        tmp = a  
        a = tmp #fault should be a=b  
        b = tmp  
  
    if a > c:  
        tmp = a  
        a = c  
        c = tmp  
  
    if b > c:  
        tmp = b  
        b = c  
        c = tmp  
  
    if a + b <= c:  
        return TriangleType.INVALID  
    elif a == b and b == c:  
        return TriangleType.EQUILATERAL  
    elif a == b or b == c:  
        return TriangleType.ISOSCELES  
    else:  
        return TriangleType.SCALENE
```

Harder Oracles

MEGAHIT Assemble Reads with MEGAHIT v1.1.1
Assemble metagenomic reads using the MEGAHIT assembler.

Run Configure Job Status Result

Input Objects

Read Library

Parameters (5 advanced parameters showing) hide advanced

Parameter preset

--min-count

--k-min 1 ≤127

--k-max 1 ≤255

--k-step 1 ≤28

--k-list

--min-contig-len 300 2000

Output Objects

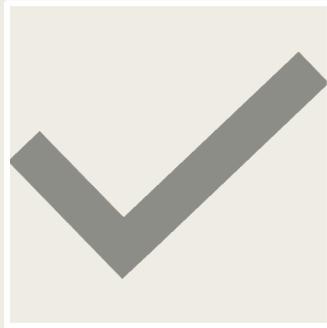
Output Assembly name



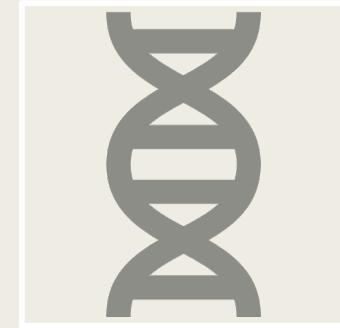
Making Oracles Hard

- Results may differ by small **epsilons** (due to rounding)
- Expected result **may not be computable** without program
- May have **time series** results
- Takes a long time to manually compute each oracle (even when we can)
- Programs may be **stochastic** (or flaky)

Some Techniques



Differential testing



Metamorphic testing

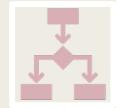
Summary of Module I: Overview



Motivation



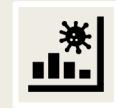
What to test



Types of
Testing



Models



Coverage



Oracles

Future Modules

Unit Testing and integrating with continuous integration

Testing configurations and combinatorial testing

Using differential and metamorphic testing

Regression testing - prioritization and test selection