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

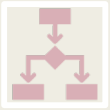
Laboratory for Variability-Aware Assurance and
Testing of Organic Programs

LaVA-OPs

Module I: Overview (cont.)

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)

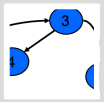
Allow us to reason about how much we have tested

The foundation for automated test generation

Example Models

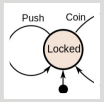
- Graphs
- Tabular
- Relational
- Grammar based
- Logic based

Graph Models

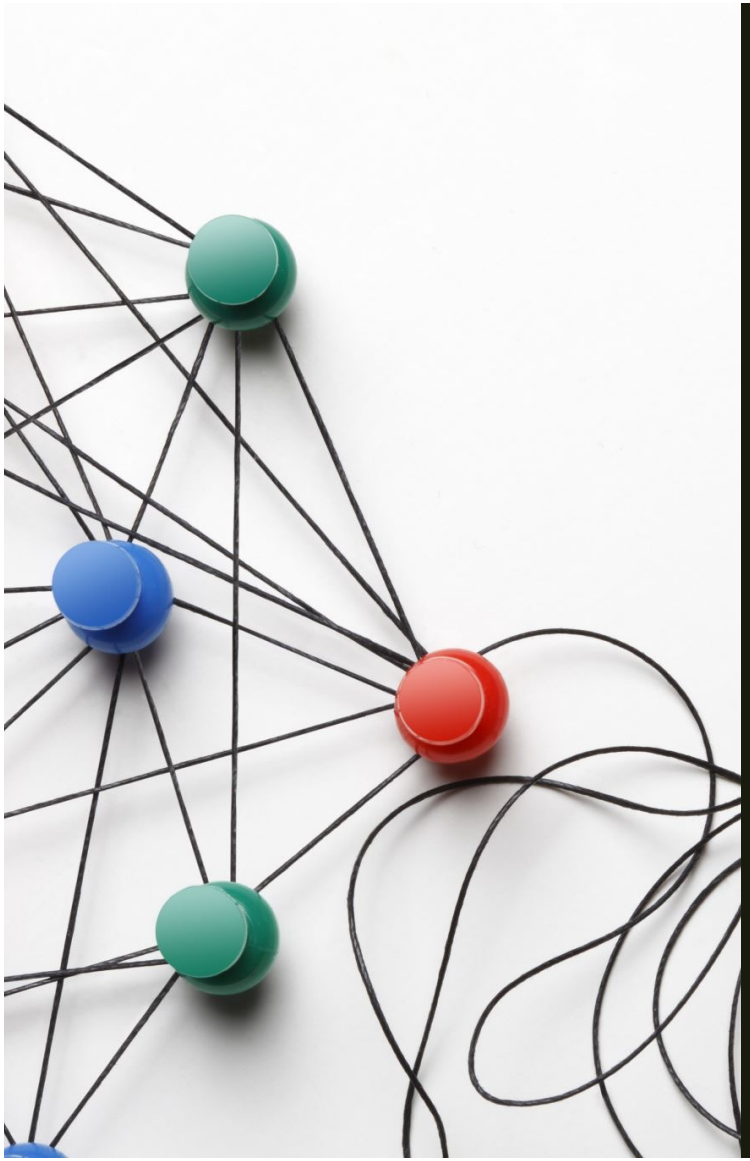


Program control flow graph

User interface



Program state machine

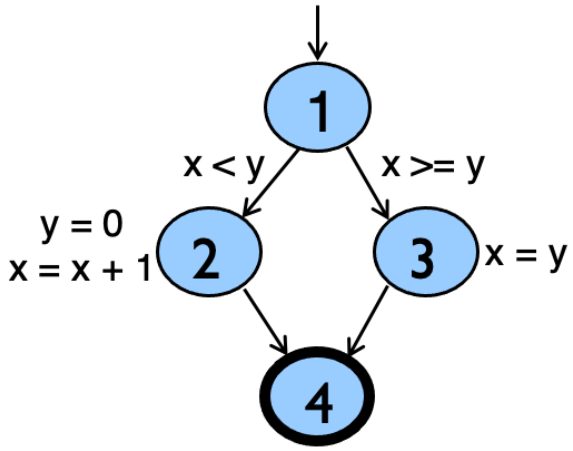


Types of Graph Coverage

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

Program Code Coverage

```
if (x < y)
{
   y = 0;
   x = x + 1;
}
else
{
   x = y;
}
```



Control flow graph

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_whiteutils.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%**

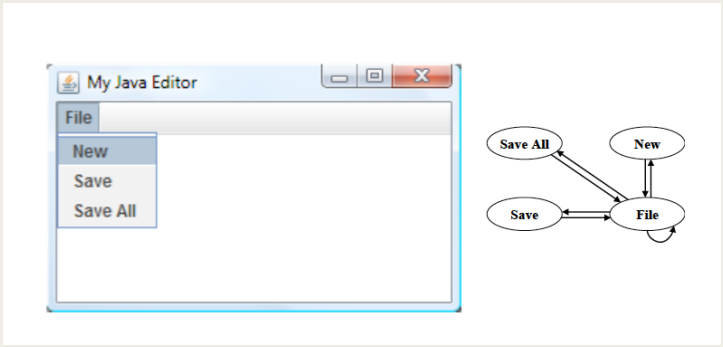
coverage.py v7.2.7, created at 2023-05-29 15:26 -0400

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.
14.         if (a > c) {
15.         int tmp = c; // original: int tmp = a;
16.             a = c;
17.             c = tmp;
18.         }
19.
20.         if (b > c) {
21.             int tmp = b;
22.             b = c;
23.             c = tmp;
24.         }
25.
26.         if (a + b <= c) {
27.             return TriangleType.INVALID;
28.         } else if (a == b && b == c) {
29.             return TriangleType.EQUILATERAL;
30.         } else if (a == b || b == c) {
31.             return TriangleType.ISOSCELES;
32.         } else {
33.             return TriangleType.SCALENE;
34.         }
35.
36.     }
37. }
```

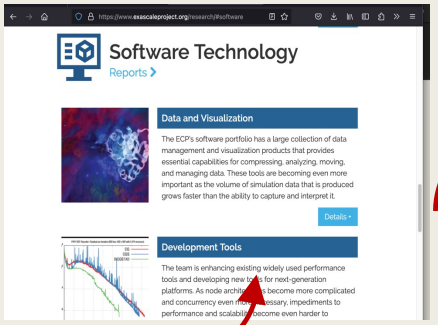
Example tools: jacoco, coverage.py, gcov

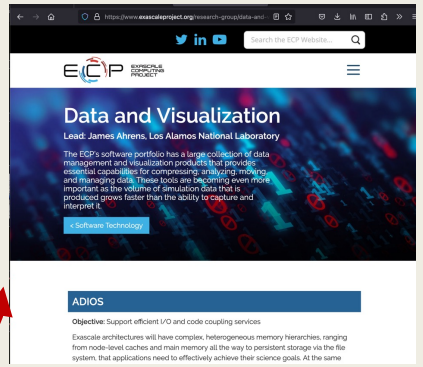
Interface (graph) Coverage

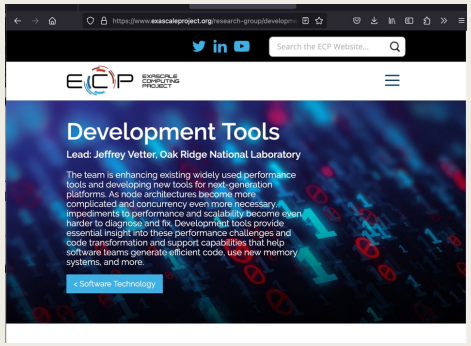


GUI

Web







Other Coverage

Specification coverage

- Cover the system requirements

Interaction coverage

- Measure interactions between components
  - Pairs, n-way coverage

Module I: Overview

Motivation

What to test

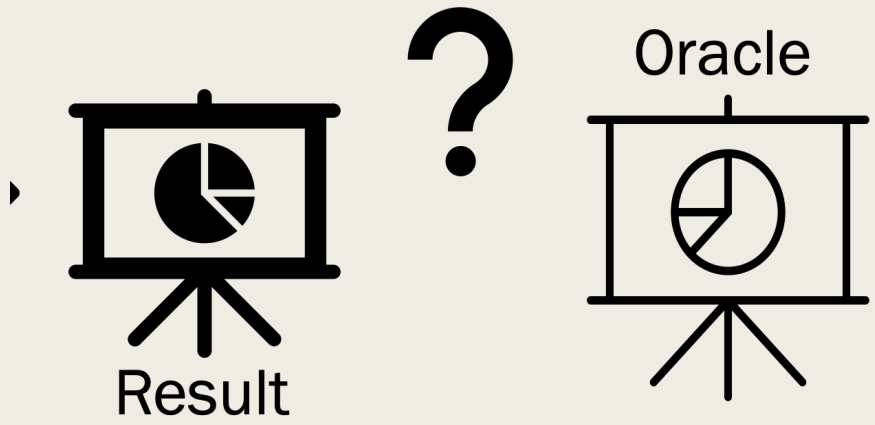
Types of Testing

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

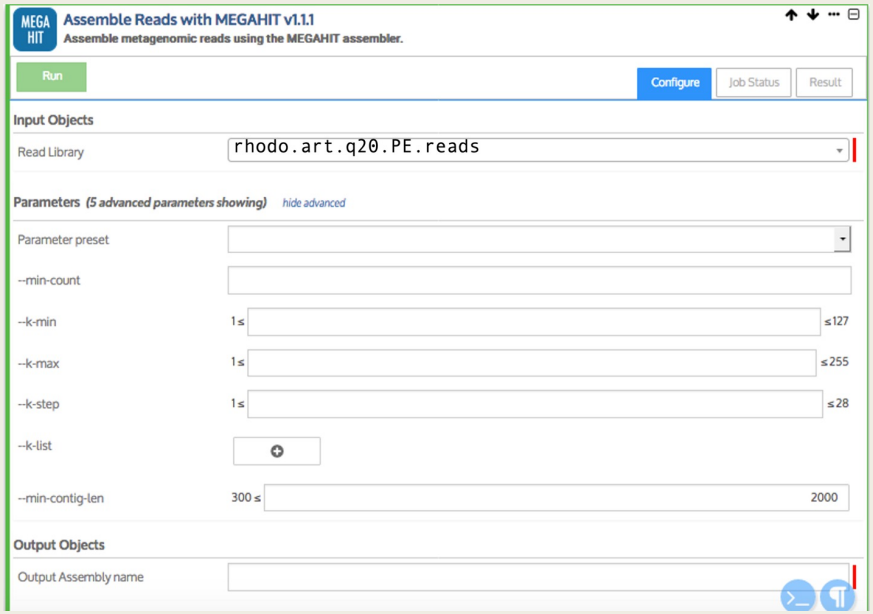
```
def classify_triangle(a, b, c):
    # Sort the sides so that a <= b <= c
    if a > b:
        tmp = a
        a = tmp   #fault should be a=b
        b = tmp

    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



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)

Examples

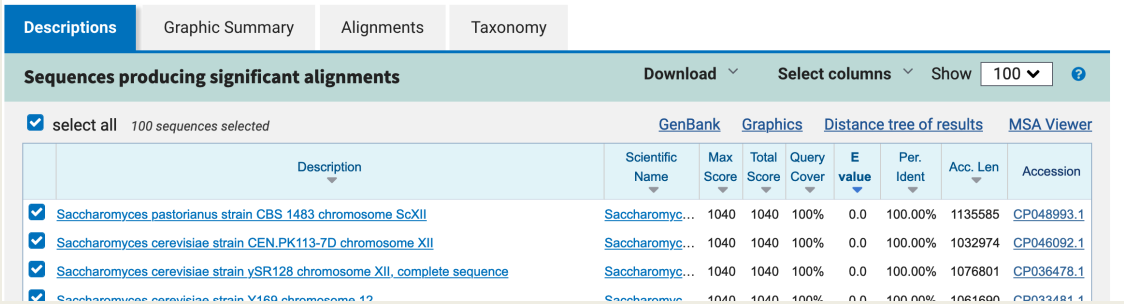
Python docs

Note: The behavior of `round()` for floats can be surprising: for example, `round(2.675, 2)` gives `2.67` instead of the expected `2.68`. This is not a bug: it's a result of the fact that most decimal fractions can't be represented exactly as a float. See Floating Point Arithmetic: Issues and Limitations for more information.

Same growth values?

Expected:	0.3569512**4**
Observed:	0.3569512**2**

Correct hits?



Some Techniques

Differential testing

Metamorphic testing

Some Techniques

Differential testing

Run same tests using different programs that have the same functionality

Some Techniques

Metamorphic testing

Define relations on sets of tests:

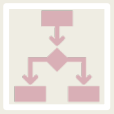
e.g. (subtraction)

A-B = C
Create A' (greater than A)
Then
A' -B = C' means C' is greater than C

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

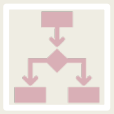
This work was supported by the Better Scientific Software Fellowship Program, funded by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration; and by the National Science Foundation (NSF) under Grant No. 2154495.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DOE or NSF.

Module I: Overview


Motivation


What to test


Types of Testing


Models


Coverage


Oracles

This work was supported by the Better Scientific Software Fellowship Program, funded by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy (DOE) Office of Science and the National Nuclear Security Administration; and by the National Science Foundation (NSF) under Grant No. 2154495.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the DOE or NSF.