
# INTRO TO SOFTWARE ENGINEERING:
# WRITING CODE THAT WORKS

Myra Cohen

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

# Today's Plan

Motivation

Overview of Software Engineering

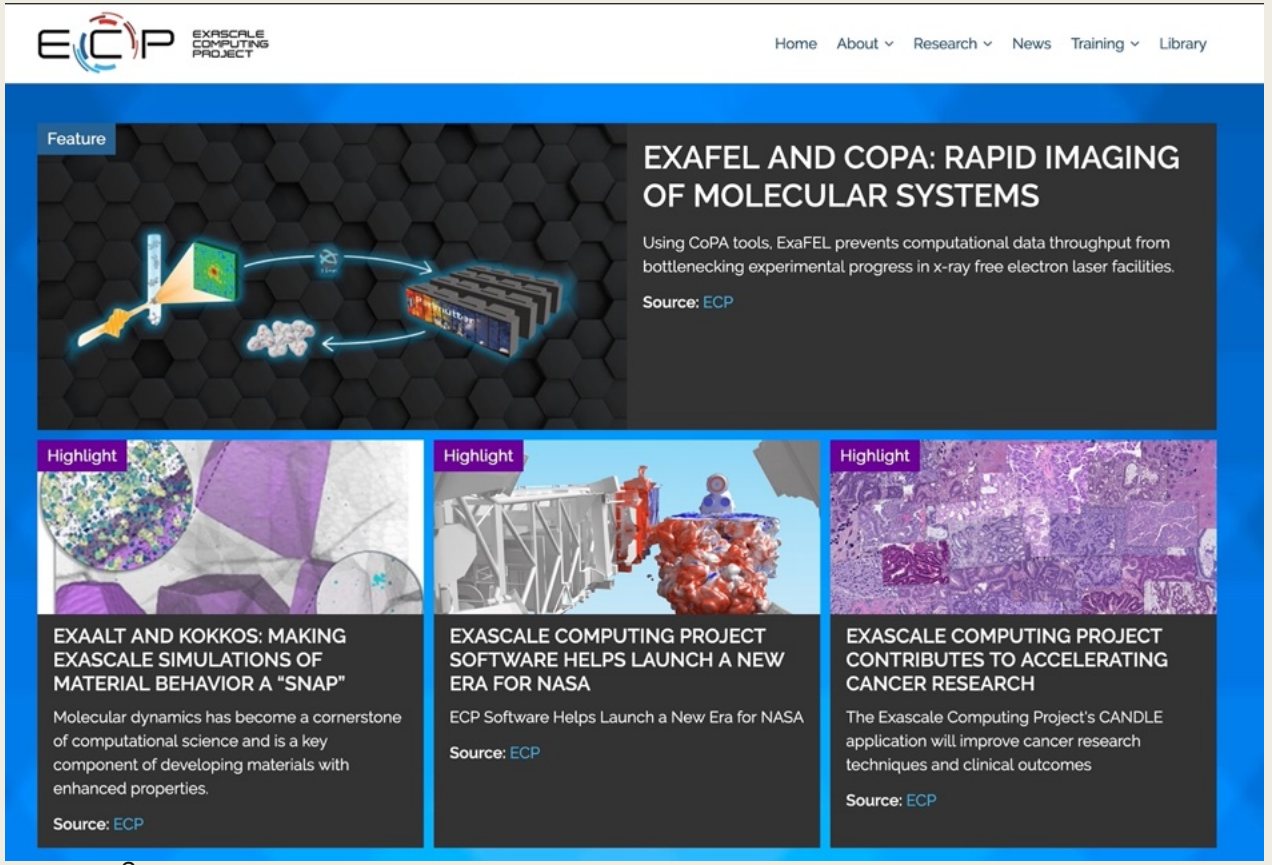
Documentation and Teams

Software Testing

Pytest

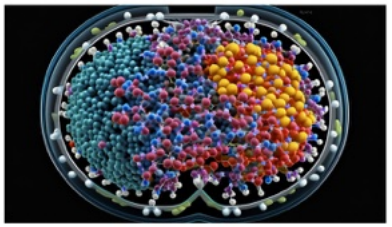
Continuous Integration

# Scientific Software is Everywhere



# Scientific Software is Everywhere


Exciting Research Projects for Interns



**Modeling of Serine Protease Substrate Catalysis In Synthetic Micelle Environment**

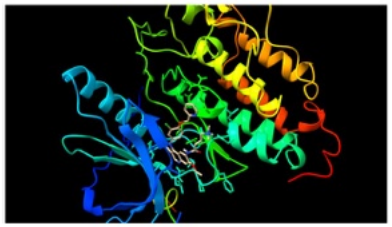
*1 min read*

Mentors: Xueyu Song, Davit Potoyan

**Automating the Fragmentation of Proteins**

*1 min read*

Mentors: Qi Li, Ryan Richard, Theresa Windus



**Machine Learning-Enhanced Computational Modeling of Metal-Protein Interactions**

*1 min read*

Mentors: Peng Xu, Mark Gordon, Qi Li, Mengdi Huai

# Yet it sometimes Fails..

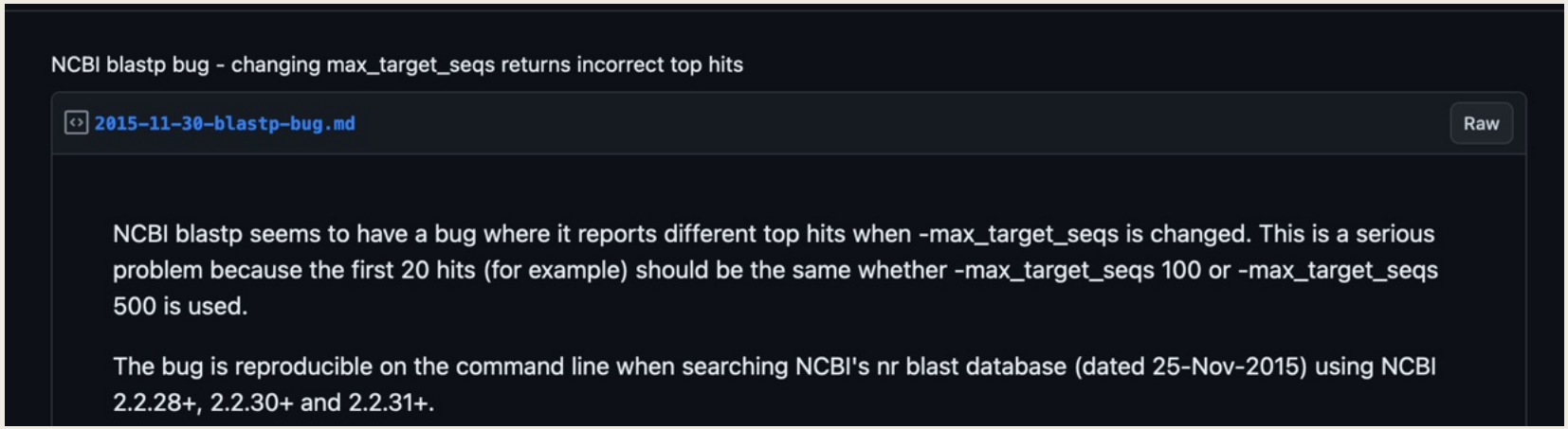
```
Error: NCBI C++ Exception:
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/objistrasnb.cpp
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/member.cpp", l
```

# Yet it sometimes Fails..

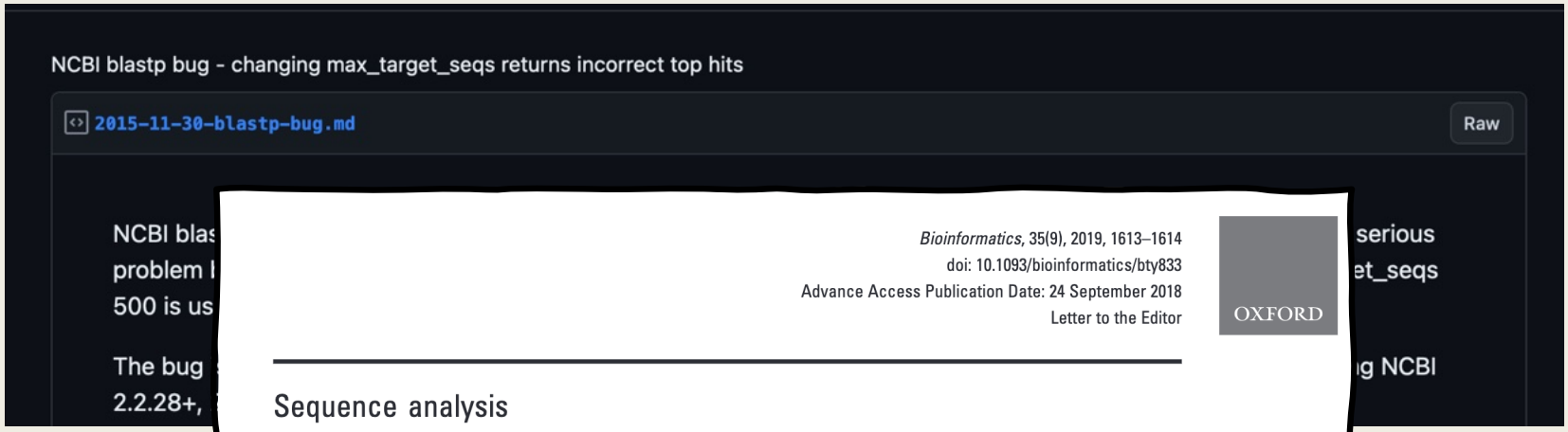
```
Error: NCBI C++ Exception:
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/objistrasnb.cpp
T0 "/tmp/BLAST/2.11.0/gompi-2020b/ncbi-blast-2.11.0+-src/c++/src/serial/member.cpp", l
```

```
ial/objistrasnb.cpp", line 499: Error: (CSerialException::eOverflow) byte 132: overf
ial/member.cpp", line 767: Error: (CSerialException::eOverflow) ncbi::CMemberInfoFur
```

# Yet it sometimes Fails..



# Yet it sometimes Fails..




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

OXFORD

Sequence analysis

## Misunderstood parameter of NCBI BLAST impacts the correctness of bioinformatics workflows

**Nidhi Shah[1], Michael G. Nute[2], Tandy Warnow [3] and Mihai Pop [1,*]**

[1]Department of Computer Science, University of Maryland College Park, MD 20742, USA, [2]Department of Statistics and [3]Department 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

# Sometimes it seems to fail..

*Bioinformatics*, 35(15), 2019, 2699–2700
doi: 10.1093/bioinformatics/bty1026
Advance Access Publication Date: 24 December 2018
Letter to the Editor

OXFORD

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

# Sometimes it seems to fail..

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.

# Cost of Poor Software Quality

- CISQ Consortium for Information & Software Quality 2020 report

  $2.08 trillion cost in United States

- 2002 National Institutes of Standards and Technology

  Up to $59 Billion per year in United States

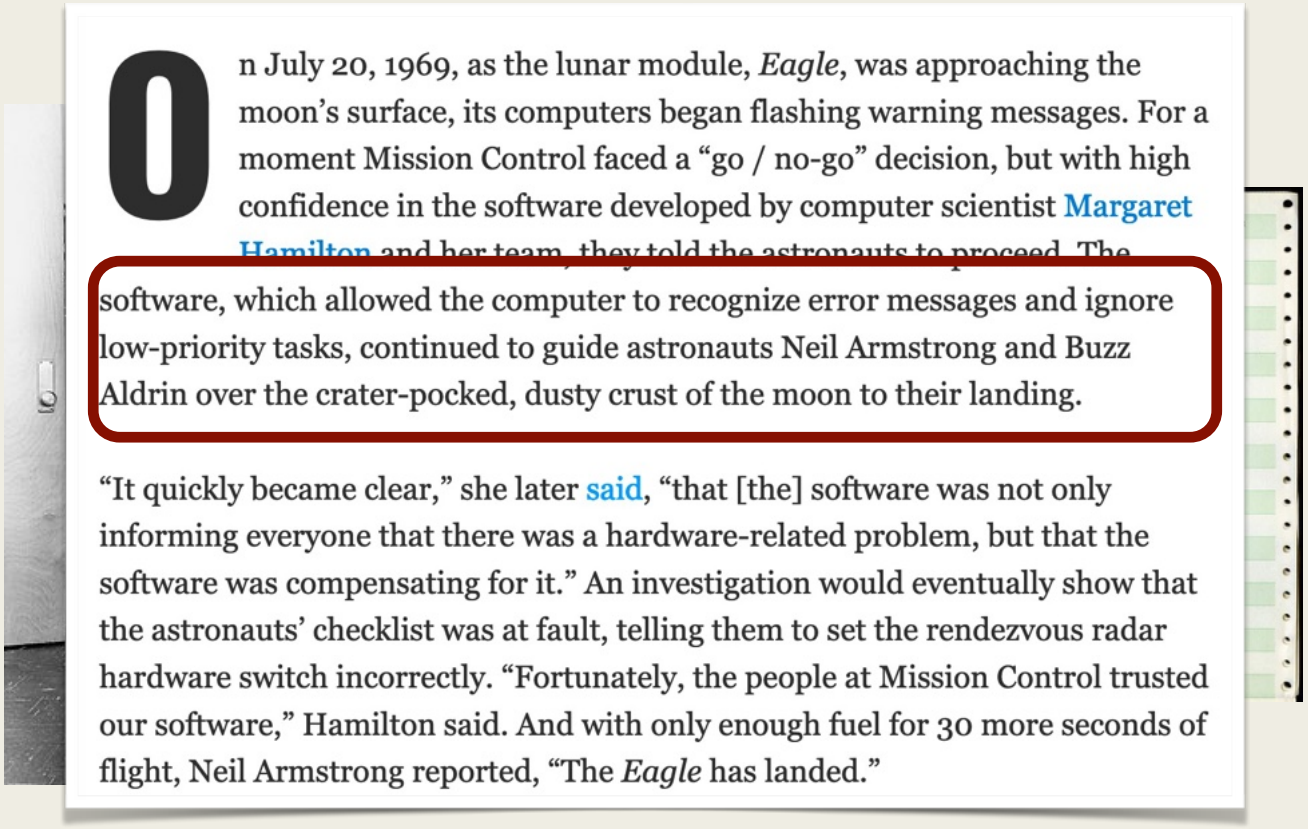
- Scientific Software

  ?

# Looking Back: 1969…

# And in Ames…

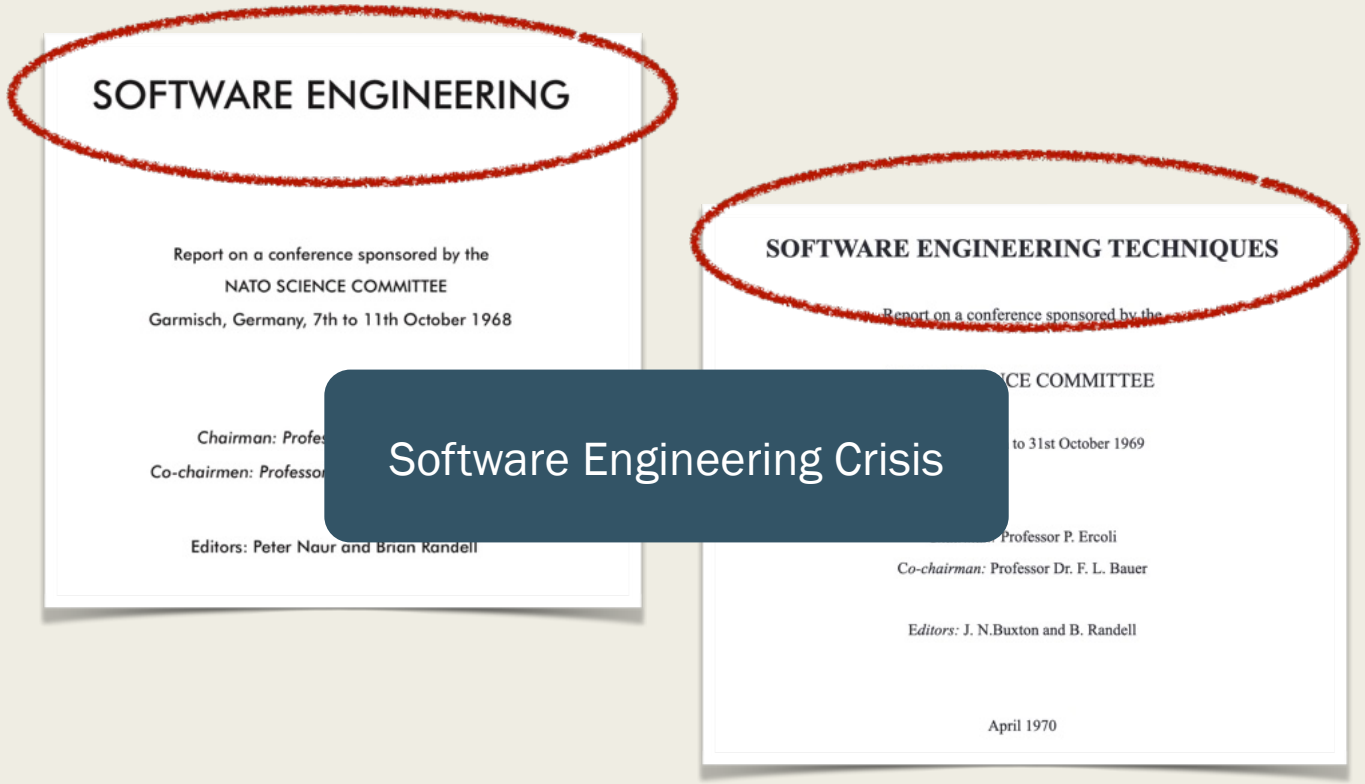
# Apollo 11 Code



On July 20, 1969, as the lunar module, *Eagle*, was approaching the moon's surface, its computers began flashing warning messages. For a moment Mission Control faced a "go / no-go" decision, but with high confidence in the software developed by computer scientist Margaret Hamilton and her team, they told the astronauts to proceed. The software, which allowed the computer to recognize error messages and ignore low-priority tasks, continued to guide astronauts Neil Armstrong and Buzz Aldrin over the crater-pocked, dusty crust of the moon to their landing.

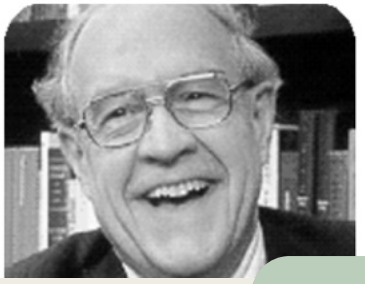
"It quickly became clear," she later said, "that [the] software was not only informing everyone that there was a hardware-related problem, but that the software was compensating for it." An investigation would eventually show that the astronauts' checklist was at fault, telling them to set the rendezvous radar hardware switch incorrectly. "Fortunately, the people at Mission Control trusted our software," Hamilton said. And with only enough fuel for 30 more seconds of flight, Neil Armstrong reported, "The *Eagle* has landed."

# Two NATO Conferences




Software Engineering Crisis

# Moving Forward: 1999



**FREDERICK ("FRED") BROOKS** DL

United States – 1999

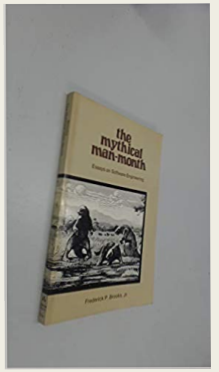
CITATION

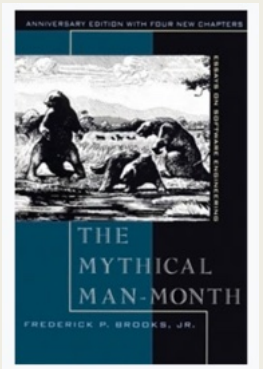
For landmark contributions to computer architecture, operating systems, and software engineering.

**"Adding manpower to a late software project, mas it later"**

**"I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared to the conceptual errors in most systems. If this is true, building software will always be hard. "**

*"There is inherently no silver bullet."*




## Accidental vs. Essential Complexity

# Today's Plan



Motivation

Overview of Software Engineering

Documentation and Teams



Software Testing

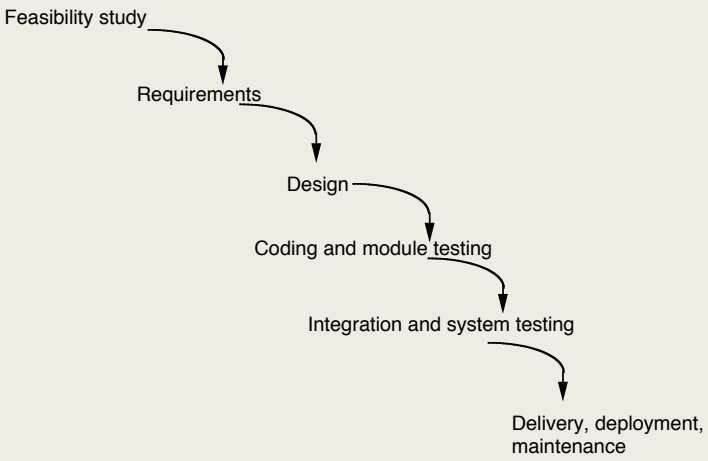
Pytest

Continuous Integration

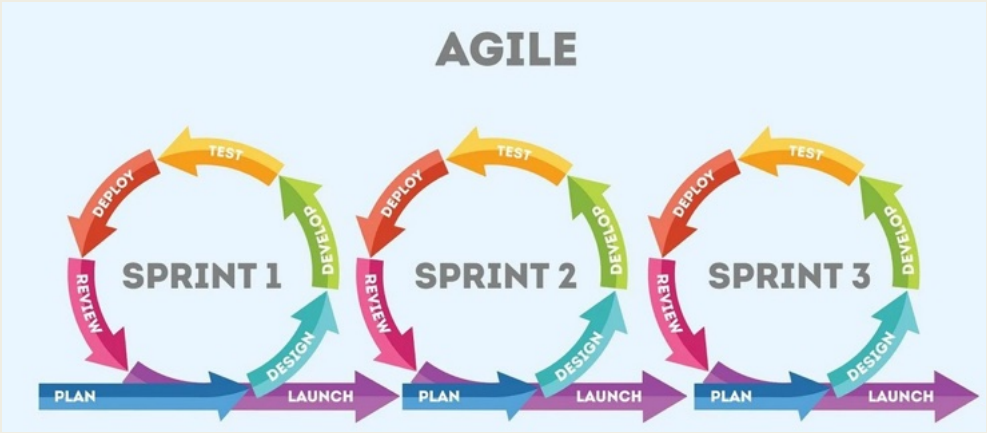
# Key Aspects of Software Engineering

- Managed software process
- Requirements
- Architecture and modularization
- Team development
- Software Testing
- Tools and Programming Environments

# Software Process




Waterfall model
works well in easy to define/traditional systems




Better for Exploratory Systems

# Requirements

- What should my system do?
  - *Natural language is not formal*
  - *Often build models or represent in logic*
  - *Includes both inputs and expected outputs*

# The “Triangle Program”

- This has become a classic example
  - *We want to write a program that reads three integers which represent the lengths of the sides of a triangle. The program displays a message that states whether the triangle is scalene, isosceles, equilateral or invalid.*

*Let’s discuss….*

# Running Example Program

```
mcohen>python triangle.py 3 4 5
This is a Scalene Triangle

mcohen>python triangle.py 3 3 5
This is an Isosceles Triangle

mcohen>python triangle.py 4 4 4
This is an Equilateral Triangle

mcohen>python triangle.py 2 2 4
This is an Invalid Triangle

mcohen>python triangle.py -1 2 4
This is an Invalid Triangle
```

# Software Architecture

- Model key components, connectors, dependencies…
- Identifies ‘smells’ – bad patterns
- Helps with software maintenance, testability,…

SonarQube cloud

# Working in Teams

- Version Control
  - *Github*
- Communication
  - *More people requires more communication*
- Documentation
  - *Important part of any project*
  - *Tools can help*

# Docstrings

## General rules

Docstrings must be defined with three double-quotes. No blank lines should be left before or after the docstring. The text starts in the next line after the opening quotes. The closing quotes have their own line (meaning that they are not at the end of the last sentence).

On rare occasions reST styles like bold text or italics will be used in docstrings, but is it common to have inline code, which is presented between backticks. The following are considered inline code:

- The name of a parameter
- Python code, a module, function, built-in, type, literal... (e.g. `os`, `list`, `numpy.abs`, `datetime.date`, `True`)
- A pandas class (in the form `:class:`pandas.Series``)
- A pandas method (in the form `:meth:`pandas.Series.sum``)
- A pandas function (in the form `:func:`pandas.to_datetime``)

```
def add_values(arr):
    """
    Add the values in ``arr``.

    This is equivalent to Python ``sum`` of :meth:`pandas.Series.sum`.

    Some sections are omitted here for simplicity.
    """
    return sum(arr)
```

```
def add(num1, num2):
    """
    Add up two integer numbers.

    This function simply wraps the ``+`` operator, and does not
    do anything interesting, except for illustrating what
    the docstring of a very simple function looks like.

    Parameters
    ----------
    num1 : int
        First number to add.
    num2 : int
        Second number to add.

    Returns
    -------
    int
        The sum of ``num1`` and ``num2``.

    See Also
    --------
    subtract : Subtract one integer from another.

    Examples
    --------
    >>> add(2, 2)
    4
    >>> add(25, 0)
    25
    >>> add(10, -10)
    0
    """
    return num1 + num2
```

# Today's Plan



Motivation



Overview of Software Engineering

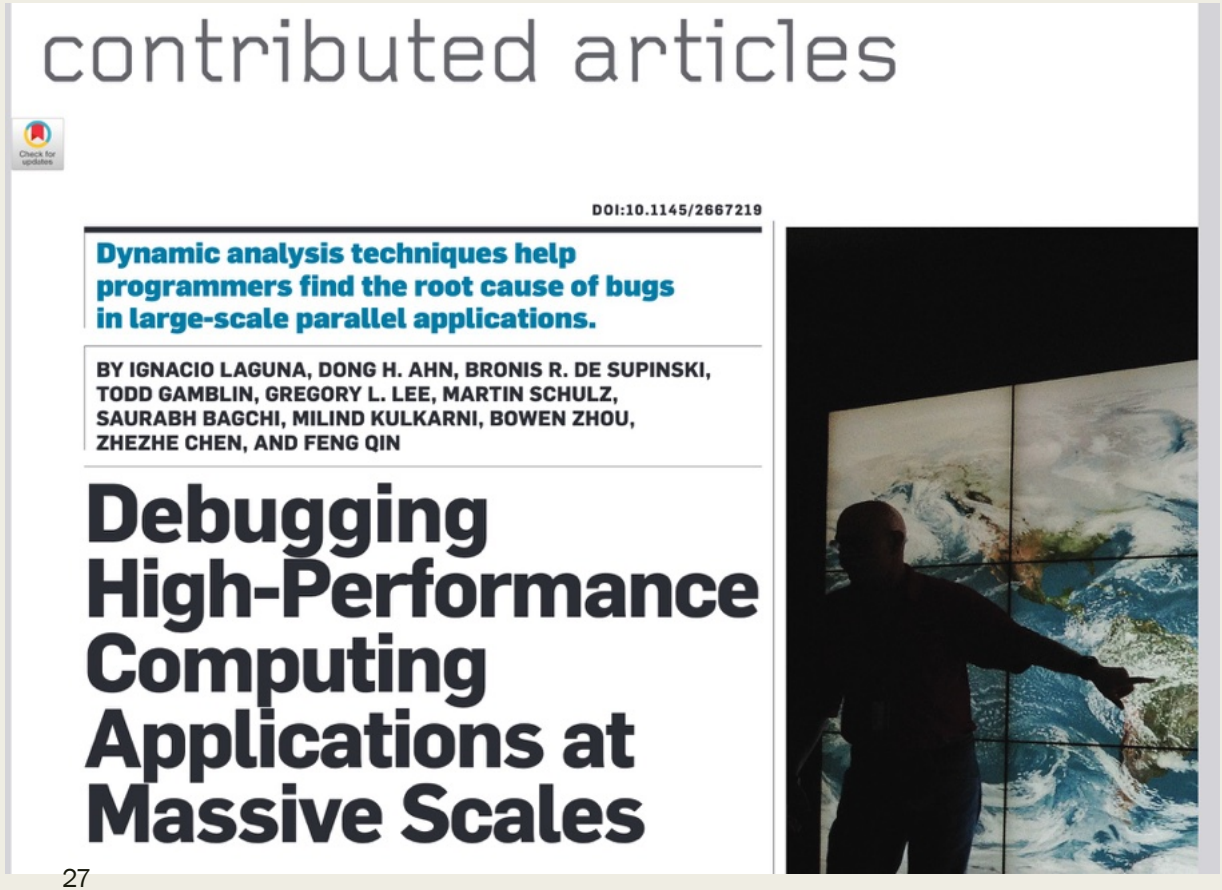
Documentation and Teams

Software Testing

Pytest

Continuous Integration

# Why Test Software?



contributed articles

DOI:10.1145/2667219

**Dynamic analysis techniques help programmers find the root cause of bugs in large-scale parallel applications.**

BY IGNACIO LAGUNA, DONG H. AHN, BRONIS R. DE SUPINSKI, TODD GAMBLIN, GREGORY L. LEE, MARTIN SCHULZ, SAURABH BAGCHI, MILIND KULKARNI, BOWEN ZHOU, ZHEZHE CHEN, AND FENG QIN

**Debugging High-Performance Computing Applications at Massive Scales**

# 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-

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

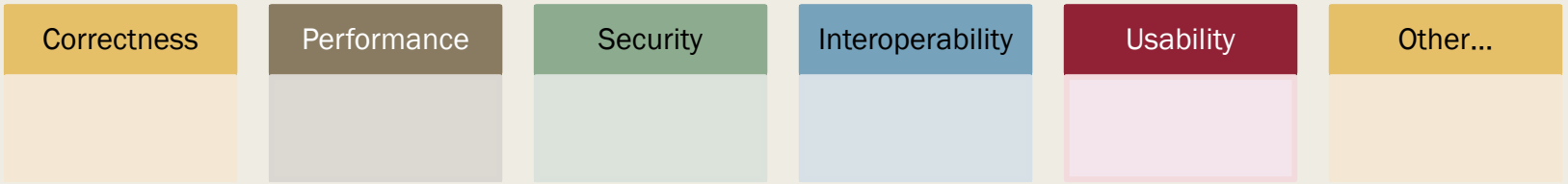
# Why Test Software?


When hackers leaked thousands of e-mails from the Climatic Research Unit (CRU) at the University of East Anglia in Norwich, UK, last year, global-warming sceptics pored over the documents for signs that researchers had manipulated data. No such evidence emerged, but the e-mails did reveal another problem — one described by a CRU employee named "Harry", who often wrote of his wrestling matches with wonky computer software.

"Yup, my awful programming strikes again," Harry lamented in one of his notes, as he attempted to correct a code analysing weather-station data from Mexico.

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

# What Should We Test?



# What Should We Test?

| Correctness | Performance | Security | Interoperability | Usability | Other… |
| --- | --- | --- | --- | --- | --- |
| | | | | | • *Extensibility*<br>• *Modularity* |

# What Should We Test?

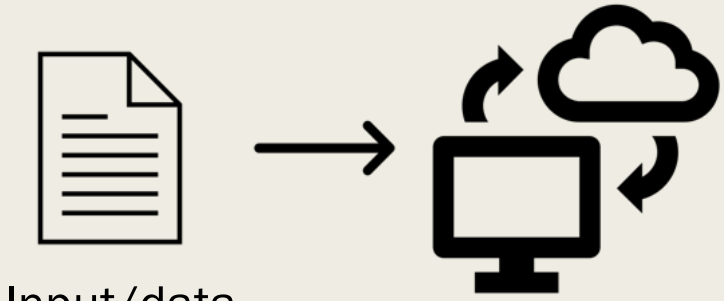
| Correctness | Performance | Security | Interoperability | Usability | Other… |
|---|---|---|---|---|---|
| | | | | | • *Extensibility*<br>• *Modularity* |

Our focus will be on correctness and interoperability

# What is Testing

Input/data

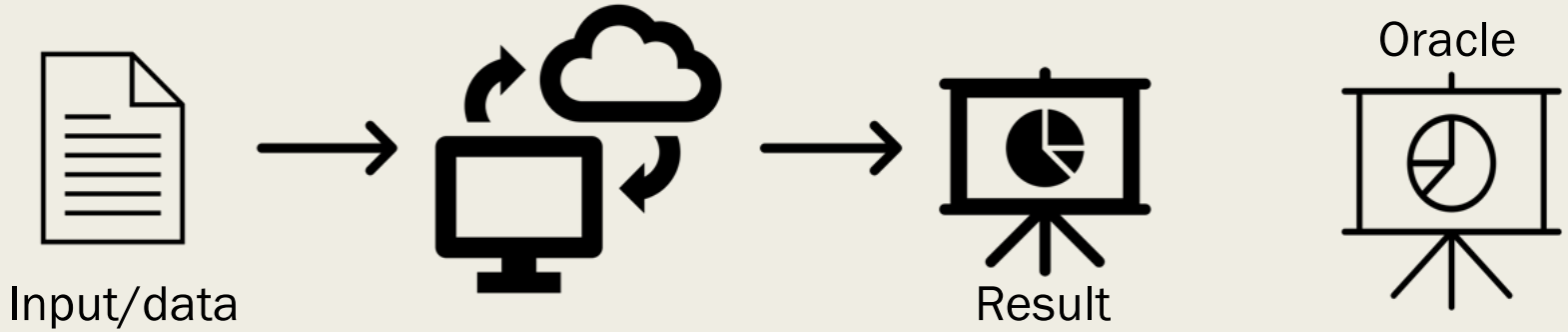
# What is Testing



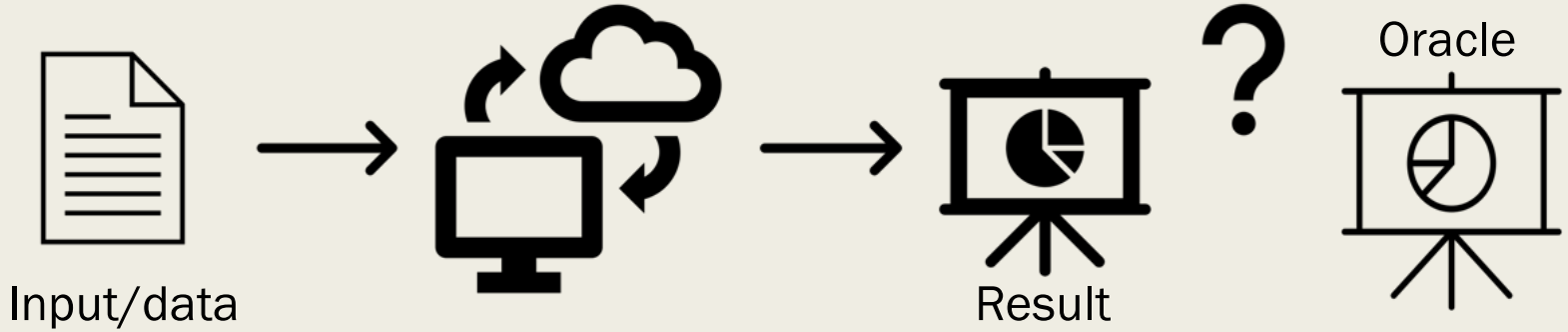
Input/data

# What is Testing

# What is Testing



# What is Testing



# Limitations

Testing can only show the presence of faults.
It cannot determine their absence.

*Edsger W. Dijkstra*

# Challenge

- 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

```python
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
```

# Tests Can Miss Faults

```python
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 (scalene)
   - doesn't reach fault

# Tests Can Miss Faults

```python
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
    5,5,1

    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 (scalene)
   - doesn't reach fault

2. Test Case 5, 1, 1 (invalid)
   - reaches fault and infects
   - reveals (returns isosceles)

# Tests Can Miss Faults

```python
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
    2,2,-1

    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 (scalene) ✗
   - doesn't reach fault
2. Test Case 5, 1, 1 (invalid) ✓
   - reaches fault and infects
   - reveals (returns isosceles)
3. Test Case 2, 1, -1 (invalid) ✗
   - reaches fault and infects
   - Doesn't propagate (2, 2,-1) is still 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


Other testing focuses on the system specifications and overall program behavior

# 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)

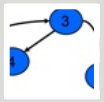
Allow us to reason about how much we have tested

The foundation for automated test generation

# Example Models

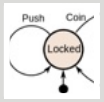
- Graphs
- Tabular
- Relational
- Grammar based
- Logic based

# Graph Models

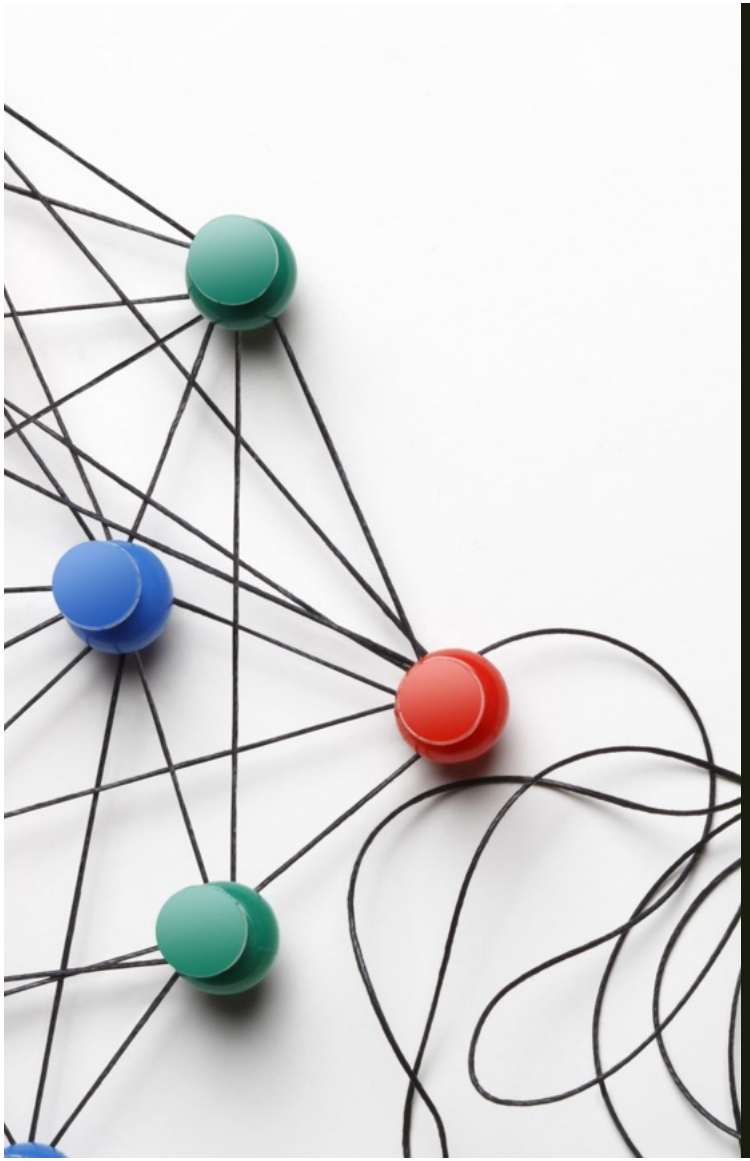



Program control flow graph

User interface




Program state machine

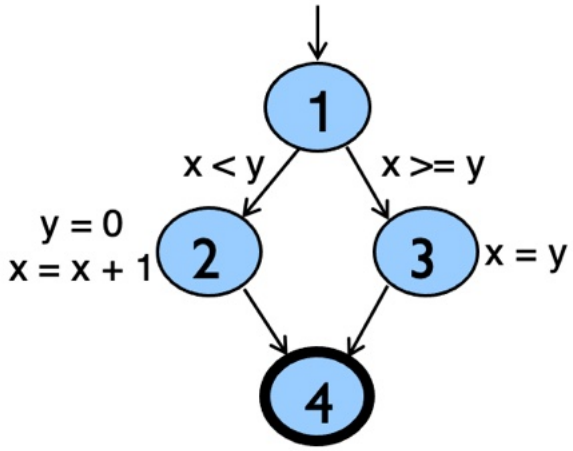


# Types of Graph Coverage

- 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

# Today's Plan



Motivation



Overview of Software Engineering

Documentation and Teams

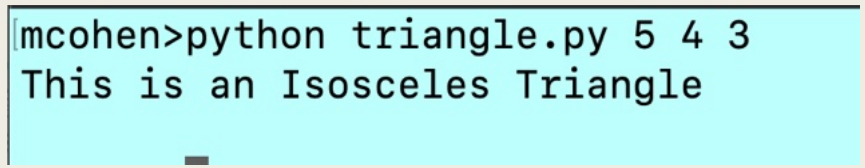


Software Testing

Pytest

Continuous Integration

```
from enum import Enum
import sys


class TriangleType(Enum):
    INVALID, EQUILATERAL, ISOSCELES, SCALENE = 0, 1, 2, 3


def classify_triangle(a, b, c):

    # Sort the sides so that a <= b <= c
    if a > b:
        tmp=c
        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
```



# Pytest


- Automated unit testing framework for Python
  - *https://docs.pytest.org/en/stable/*

  *pip install -U pytest*

  *Example pytest file*

```python
#based on test file from
import time
import pytest
from bootcampexample.triangle import TriangleType, classify_triangle


def check_classification(triangles, expected_result):
    for triangle in triangles:
        assert classify_triangle(*triangle) == expected_result


def test_invalid_triangles():
    triangles = [(1, 2, 9),  (1, 1, -1)]
    check_classification(triangles, TriangleType.INVALID)


def test_equalateral_triangles():
    triangles = [(1, 1, 1), (100, 100, 100), (99, 99, 99)]
    check_classification(triangles, TriangleType.EQUILATERAL)


def test_isoceles_triangles():
    triangles = [(20, 41, 41), (41, 20, 41),]

    check_classification(triangles, TriangleType.ISOSCELES)


def test_scalene_triangles():
    triangles = [ (3, 4, 5), (3, 5, 4)]
    check_classification(triangles, TriangleType.SCALENE)


@pytest.fixture(scope="session", autouse=True)
def starter(request):
    start_time = time.time()

    def finalizer():
        print("runtime: {}".format(str(time.time() - start_time)))

    request.addfinalizer(finalizer)
```

```
mcohen>pytest
============================================ test session starts =======
platform darwin -- Python 3.12.3, pytest-8.2.1, pluggy-1.5.0
rootdir: /Users/mbcohen/research/Bootcamp/bootcampexample
configfile: pytest.ini
collected 4 items

tests/test_triangle.py ....

============================================ 4 passed in 0.02s ========
mcohen>
```

Add 5 4 4 to Isosceles
Add 5 6 3 to Scalene

```
    def test_isoceles_triangles():
        triangles = [(20, 41, 41), (41, 20, 41), (5, 4, 4)]

>       check_classification(triangles, TriangleType.ISOSCELES)

tests/test_triangle.py:25:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

triangles = [(20, 41, 41), (41, 20, 41), (5, 4, 4)], expected_result = <TriangleType.ISOSCELES: 2>

    def check_classification(triangles, expected_result):
        for triangle in triangles:
>           assert classify_triangle(*triangle) == expected_result
E           assert <TriangleType.EQUILATERAL: 1> == <TriangleType.ISOSCELES: 2>
E            +  where <TriangleType.EQUILATERAL: 1> = classify_triangle(*(5, 4, 4))

tests/test_triangle.py:9: AssertionError
_____________________________________________________________________ test_scalene_triangles _____________________________________

    def test_scalene_triangles():
        triangles = [ (3, 4, 5), (3, 5, 4), (5, 4, 3)]
>       check_classification(triangles, TriangleType.SCALENE)

tests/test_triangle.py:30:
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

triangles = [(3, 4, 5), (3, 5, 4), (5, 4, 3)], expected_result = <TriangleType.SCALENE: 3>

    def check_classification(triangles, expected_result):
        for triangle in triangles:
>           assert classify_triangle(*triangle) == expected_result
E           assert <TriangleType.ISOSCELES: 2> == <TriangleType.SCALENE: 3>
E            +  where <TriangleType.ISOSCELES: 2> = classify_triangle(*(5, 4, 3))

tests/test_triangle.py:9: AssertionError
------------------------------------------------------------------- Captured stdout teardown -------------------------------------
runtime: 0.026725053787231445
=============================================================================== short test summary info ===========================
FAILED tests/test_triangle.py::test_isoceles_triangles - assert <TriangleType.EQUILATERAL: 1> == <TriangleType.ISOSCELES: 2>
FAILED tests/test_triangle.py::test_scalene_triangles - assert <TriangleType.ISOSCELES: 2> == <TriangleType.SCALENE: 3>
============================================================================== 2 failed, 2 passed in 0.12s ========================
```

# Code Coverage

- >coverage run -m pytest
- > coverage report

```
mcohen>coverage report
Name                       Stmts   Miss  Cover
----------------------------------------------
__init__.py                    0      0   100%
tests/__init__.py              0      0   100%
tests/test_triangle.py        23      0   100%
triangle.py                   42     16    62%
----------------------------------------------
TOTAL                         65     16    75%
```

```
mcohen>coverage report --show-missing
Name                       Stmts   Miss  Cover   Missing
-----------------------------------------------------------
__init__.py                    0      0   100%
tests/__init__.py              0      0   100%
tests/test_triangle.py        23      0   100%
triangle.py                   42     16    62%   41-57, 61
-----------------------------------------------------------
TOTAL                         65     16    75%
```

# Branch Coverage

- `mcohen>coverage run -m --branch pytest`
- > coverage html

# Today's Plan



Motivation



Overview of Software Engineering

Documentation and Teams

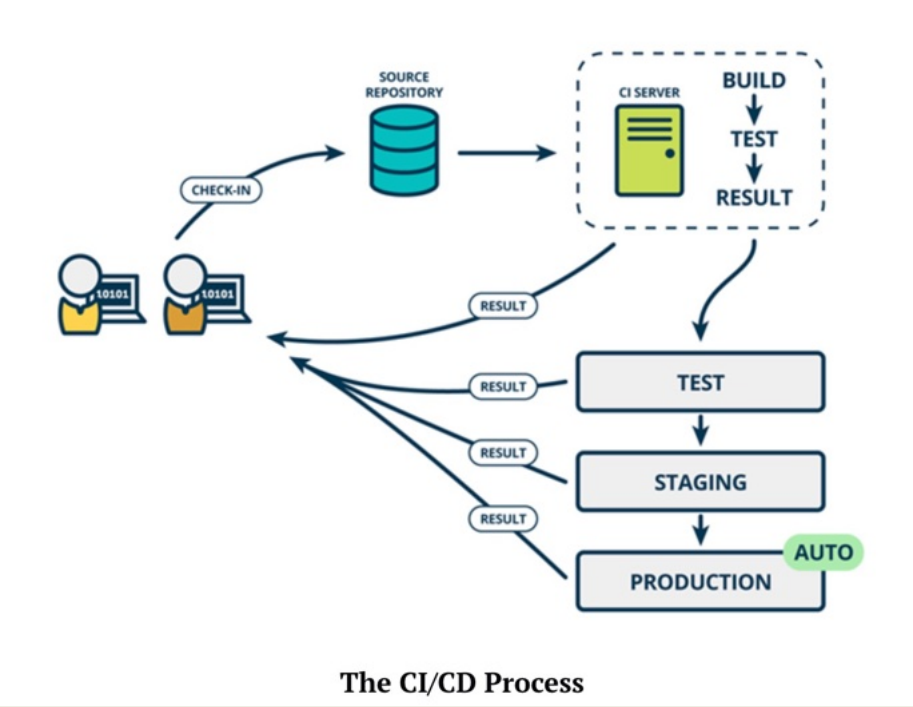


Software Testing

Pytest

Continuous Integration

# Continuous Integration/Continuous Development (CI/CD)




The CI/CD Process

# Continuous Integration

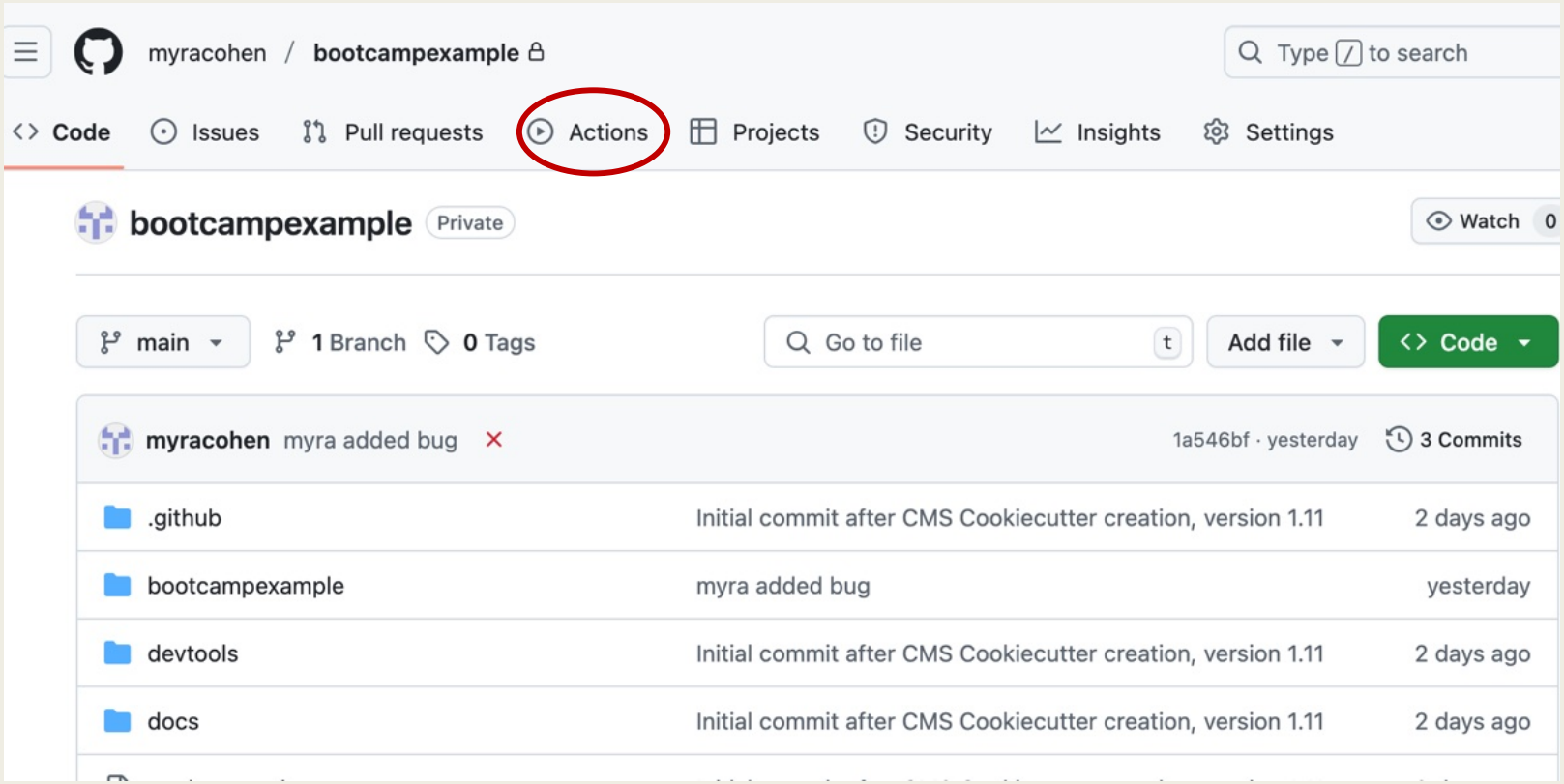
A *continuous integration server* rebuilds the system, returns, and re-verifies tests whenever *any* update is checked into the repository

A *continuous integration server* doesn't just run tests, it decides if a modified system is still correct

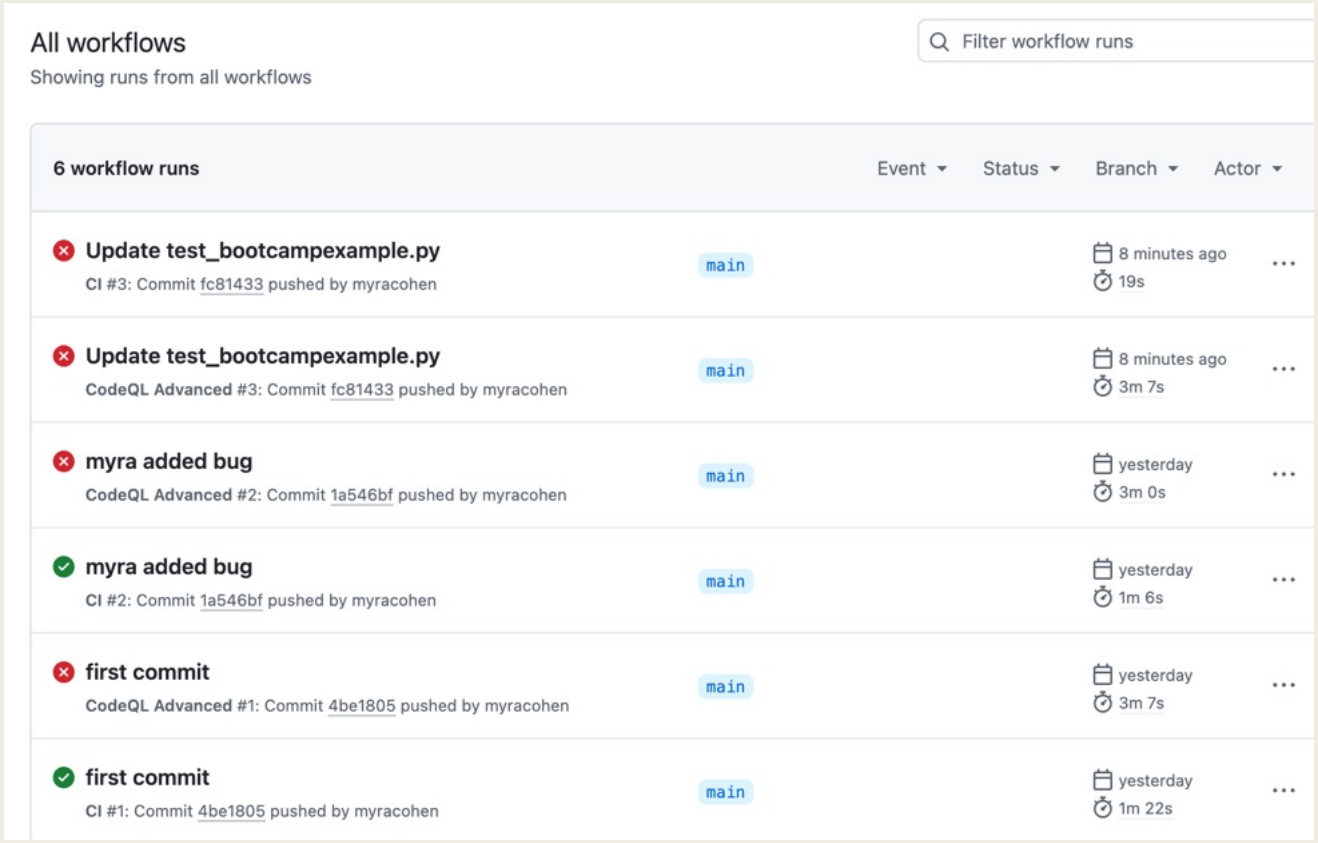
- Mistakes are caught earlier
- Other developers are aware of changes early
- The rebuild and re-verify must happen as soon as possible

Ammann, Offutt (Introduction to Software Testing, 2016)

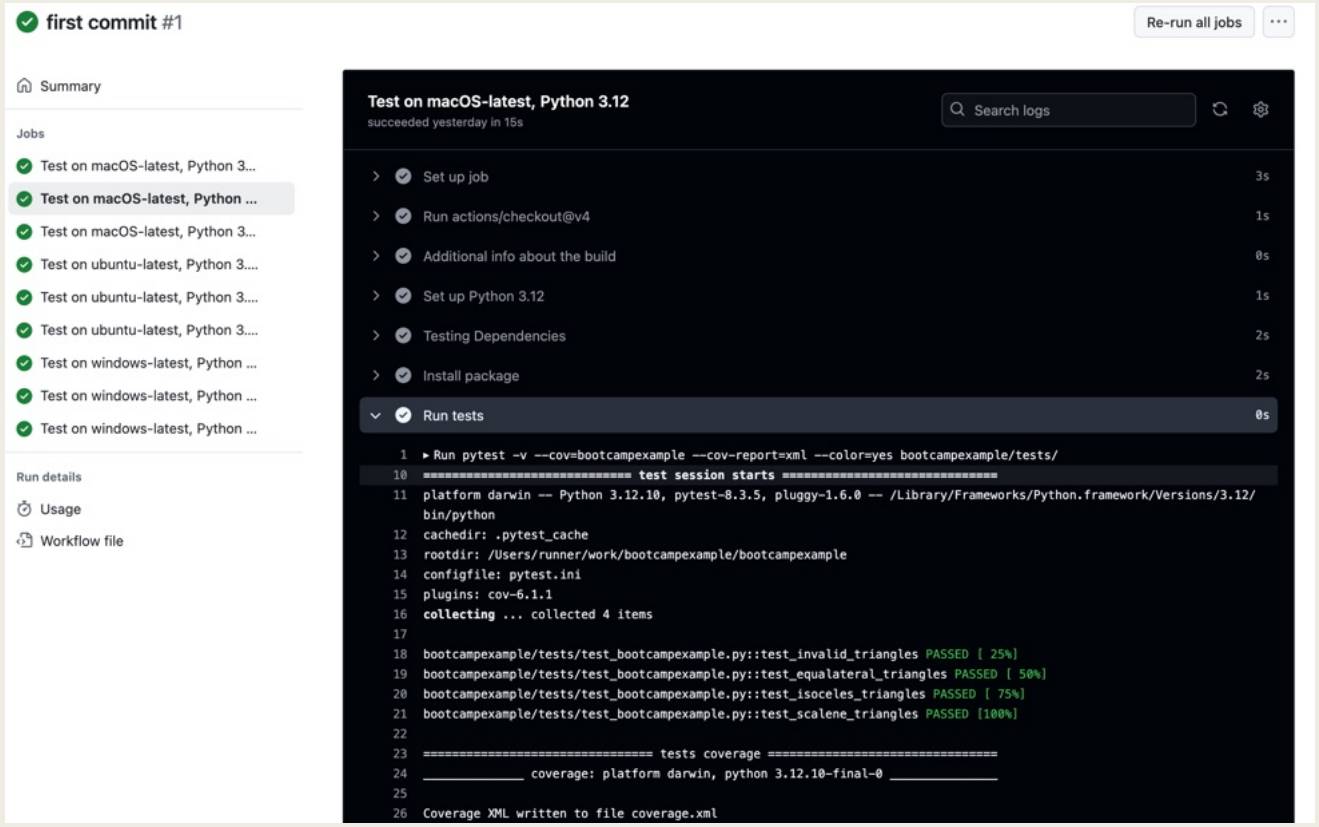
# Cookie Cutter uses Github Actions



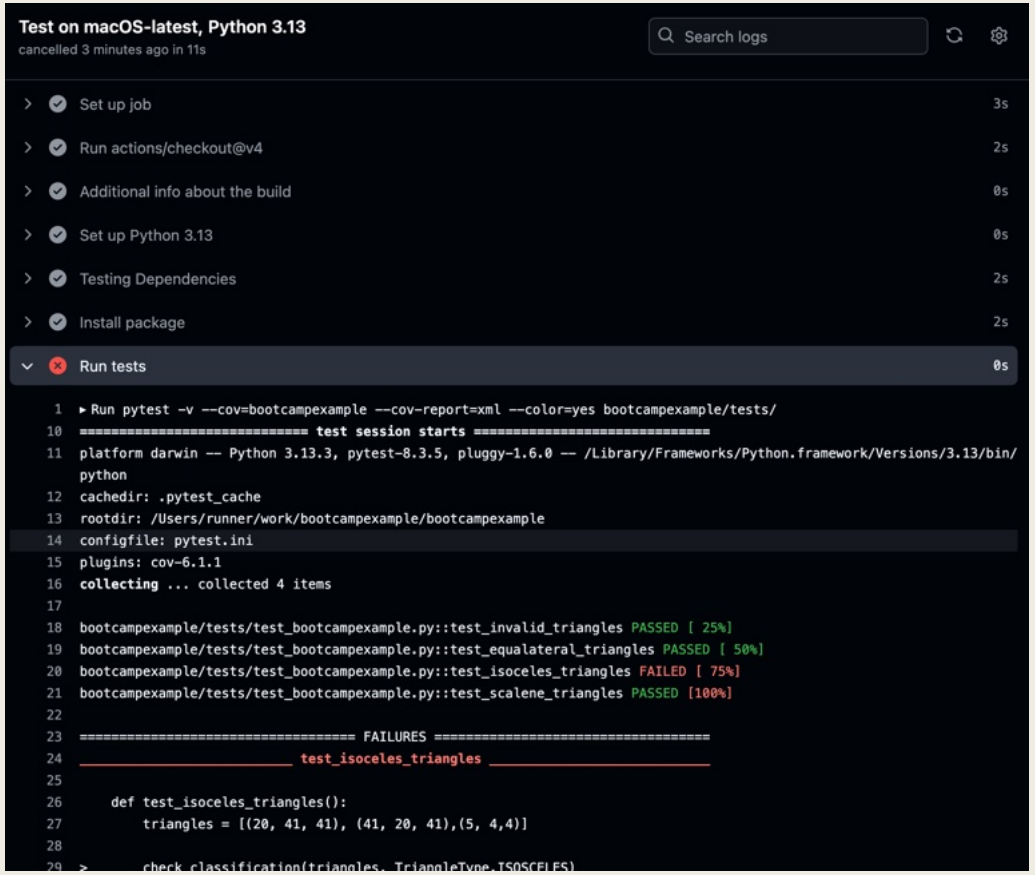
# Cookie Cutter uses Github Actions



# Cookie Cutter uses Github Actions



# Cookie Cutter uses Github Actions



# Summary

Motivation

Overview of Software Engineering

Documentation and Teams

Software Testing

Pytest

Continuous Integration

*This work was supported by the Office of Biological and Environmental Research (BER) in the U.S. Department of Energy (DOE) Office of Science award DE-SC0025510 and by NSF awards #1909688 and #2435255.*

*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.*