ARA Public Launch Workshop Day 2

AN INTRODUCTION TO SOFTWARE TESTING

Myra Cohen

https://cs.iastate.edu/mcohen

Iowa State University of Science and Technology

better scientific software

Laboratory for Variability-Aware Assurance and Testing of Organic Programs

LaVA-OPs

1

Overview

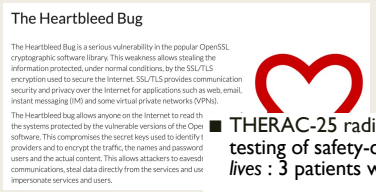
Motivation | What to test | Types of Testing | Models | Coverage | Oracles

2

Software is Everywhere

3

When it Fails...



- THERAC-25 radiation machine : Poor testing of safety-critical software can cost *lives* : 3 patients were killed
- NASA's Mars Lander: September 1999, crashed due to a units integration fault

4

Toyota Acceleration

Carnegie Mellon

Toyota Case: Single Bit Flip That Killed

Junko Yoshida
10/25/2013 03:35 PM EDT

During the trial, embedded systems experts who reviewed Toyota's electronic throttle source code testified that they found Toyota's source code defective, and that it contains bugs -- including bugs that can cause unintended acceleration.

"We did a few things that NASA apparently did not have time to do," Barr said. For one thing, by looking within the real-time operating system, the experts identified "unprotected critical variables." They obtained and reviewed the source code for the "sub-CPU," and they "uncovered gaps and defects in the throttle fail safes."

The experts demonstrated that "the defects we found were linked to unintended acceleration through vehicle testing," Barr said. "We also obtained and reviewed the source code for the black box and found that it can record false information about the driver's actions in the final seconds before a crash."

Stack overflow and software bugs led to memory corruption, he said. And it turns out that the crux of the issue was these memory corruptions, which acted "like ricocheting bullets."

Barr also said more than half the dozens of tasks' deaths studied by the experts in their experiments "were not detected by any fail safe."

14

Bookout Trial Reporting

http://www.eetimes.com/document.asp?doc_id=1319903&page_number=1 (excerpts)

"Task X death in combination with other task deaths"

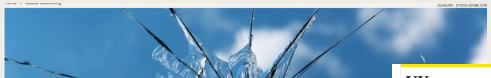
5

Customs Disruptions (Aug 2019)

- Customs processing at big US International Airports, came to a stop

6

And in Wireless Networks



NEWS

Software failure paralyses O2's 4G network

O2 is racing to fix a major outage on its 4G network that has left millions of mobile subscribers without access to data services

By Alex Scroxton, Security Editor

Published: 06 Dec 2018 13:45

Mobile network operator O2 has blamed a problem with a third-party software installation for a nationwide network outage that has left millions of subscribers unable to access 4G data services on their smartphones.

UK emergency alert test: Three looking into why users failed to get text

Network's users report en masse that SMS, accompanied by piercing 10-second tone, never arrived

UK emergency alert test: live updates

Some users received the alert up to a minute early. Photograph: Leon Neal/Getty Images

The mobile network Three said it was investigating why many of its users failed to receive an emergency alert from the government, the first

7

Why Test Software?

NCBI blastp bug - changing max_target_seqs returns incorrect top hits

2015-11-30-blastp-bug.md — Raw

NCBI blas… problem l… 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

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

…serious …et_seqs …g NCBI

8

Why Test Software?


contributed articles

DOI:10.1145/1330037

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-

9

Why Test Software?


contributed articles

DOI:10.1145/1330037

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-

10

Consequences

- National Institutes of Standards (NIST)– up to $59 Billion per year – could be cut in ½ with better testing
- Government accountability report (GAO) report DOD loses $8 Billion annually due to rework [2004]

11

Consequences Cont.

- Huge losses due to web application failures and inefficiencies
  - *Financial services : $6.5 million per hour (just in USA!)*
  - *Credit card sales applications : $2.4 million per hour (in USA)*
  - *2007 : Symantec says that most security vulnerabilities are due to faulty software*

12

This is old news…

"an analyzing process must equally have been performed in order to furnish the Analytical Engine with the necessary operative data; and that herein may also lie a possible source of error. Granted that the actual mechanism is unerring in its processes, the cards may give it wrong orders." – Ada, Countess Lovelace (notes on Babbage's Analytical Engine)

"It has been just so in all of my inventions. The first step is an intuition, and comes with a burst, then difficulties arise—this thing gives out and *[it is]* then that 'Bugs'—as such little faults and difficulties are called—show themselves and months of intense watching, study and labor are requisite. . ." – Thomas Edison

13

Approaches to Avoid Failures

Static

Dynamic

14

Static Techniques

Analyze code without running the program

- Software inspections
  - *Analysis of source (and docs) against a checklist of common and historical defects*
- Software reviews
  - Meetings of project personnel to discuss aspects of code/documents. May include stakeholders

15

Static Techniques

- Formal Verification
  - *(1) Model checking*
    - Takes a model of the program (description of functional requirements) and properties that the program is supposed to satisfy
    - Searches state-space to uncover violations of this property
  - *(2) Theorem proving*

16

Dynamic Technique

- Software Testing

17

What Should We Test?

18

What Should We Test?

19

What Should We Test?

20

What Should We Test?

Correctness	Performance	Security	Interoperability	Usability	Other…
					• *Extensibility* • *Modularity*

Our focus will be on correctness and interoperability
In networking software there is an importance focus on performance

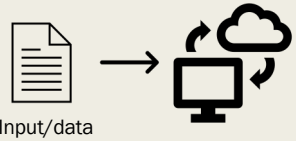
21

What is Testing

Input/data

22

What is Testing

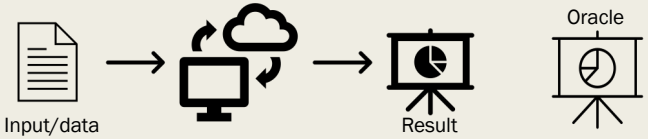


23

What is Testing

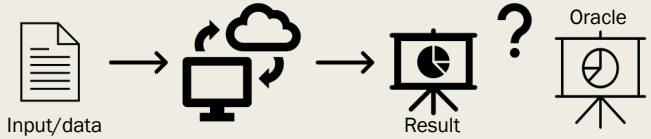
24

What is Testing



25

What is Testing



26

Some Definitions

- Software Fault: A static defect in the software
- Software Error: An incorrect internal state that is the manifestation of some fault
- Software Failure: External, incorrect behavior with respect to the requirements or other description of the expected behavior

27

Example

- Patient gives doctor list of symptoms (failures)
- Doctor tries to diagnose root cause or disease (fault)
- Doctor may look for internal irregularities (blood work, etc.) (error)

28

Some Definitions

- Test Case: A single input to the system along with its expected output
- Test Suite: A set of test cases

29

A Testers Goal

- To find faults and eliminate them as early as possible in the testing process
- NOT to prove that software works

GOAL

- *Improve software quality*
- *Improve customer's perceived experience*
- *Reduce cost*

"Testing can never prove the absence of faults – it can only prove their existence"
– E. Dijkstra

30

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

31

Test Oracles

- The expected output of a test case
- We verify that running our software on a selected test set returns expected results, or finds errors
- Each test case must include an oracle
- Determining the oracle for the test input is often a difficult and manually intensive process

33

33

Software vs. Data

- Cyber-physical systems are often data (or model driven)
- Both software and data can lead to faults
- We start by focusing on software

34

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

35

Module Overview

Motivation | What to test | Types of Testing | Models | Coverage | Oracles

36

Types of Testing

- Unit Testing
- Integration Testing
- System Testing
- Configuration Testing
- User Interface Testing
- Regression Testing

37

Types of Testing

- Unit: a developer tests their own code (Junit)
- Integration: test multiple units together (developer or tester)
- System: test full system together. Often lack source code. Performed by someone other than the developer

38

Regression Testing

39

Regression Testing

- Performed throughout program maintenance
- Each time a program is modified (to fix a fault or add new functions)
- Ensures that the previously working code has not broken (regressed)
- Expensive because it happens often

40

Other Types of Software Testing

- Performance Testing
  - *Validates whether or not the system meets specified performance criteria*
  - *This can be performance based on algorithms or it can be user performance*
- Usability Testing
  - *Evaluates the ease of using/learning the system and user documentation as well as the functionality of the system*
- Configuration Testing
  - *Failures are often due to interactions between a small number of combinations of configuration parameters*

41

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

42

Example Models

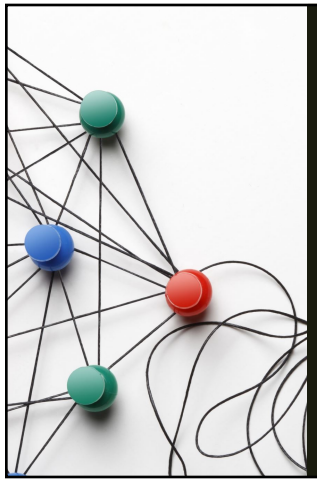
- Graphs
- Tabular
- Relational
- Grammar based
- Logic based

43

Graph Models

- Program control flow graph
- User interface
- Program state machine

44



Types of Graph Coverage

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

45

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

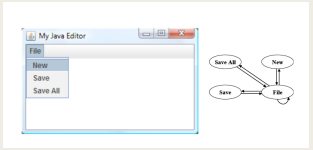
Triangle.java

```
public class Triangle {

    public enum TriangleType {
        INVALID, SCALENE, EQUILATERAL, ISOSCELES
    }

    public static TriangleType classifyTriangle(int a, int b, int c) {
        if (a > b) {
            int tmp = a;
            a = b;
            b = tmp;
        }

        if (a > c) {
            int tmp = c; // originally: int tmp = a;
            a = c;
            c = tmp;
        }

        if (b > c) {
            int tmp = b;
            b = c;
            c = tmp;
        }

        if (a + b <= c) {
            return TriangleType.INVALID;
        } else if (a == b && b == c) {
            return TriangleType.EQUILATERAL;
        } else if (a == b || b == c) {
            return TriangleType.ISOSCELES;
        } else {
            return TriangleType.SCALENE;
        }
    }
}
```

Example tools: jacoco, coverage.py, gcov

46

Interface (graph) Coverage



GUI

Web

47

Other Coverage

Specification coverage

- Cover the system requirements

Interaction coverage

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

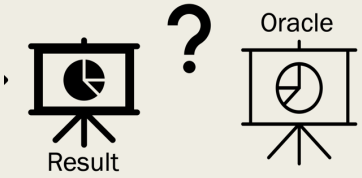
48

Module I: Overview

Motivation | What to test | Types of Testing | Models | Coverage | Oracles

49

What is the Correct Answer?



50

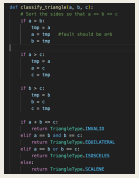
Trivial Oracles

Program crashes
Core dump
Segmentation error
Overflow
Program hangs

51

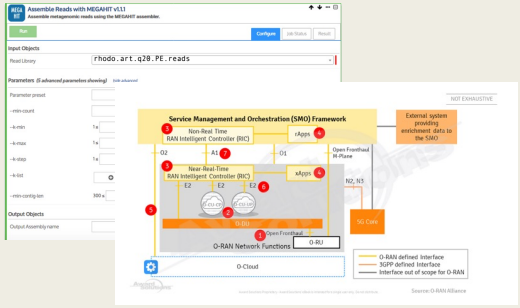
Trivial Oracles

- 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



52

Harder Oracles



53

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)

54

Some Techniques

Differential testing

Metamorphic testing

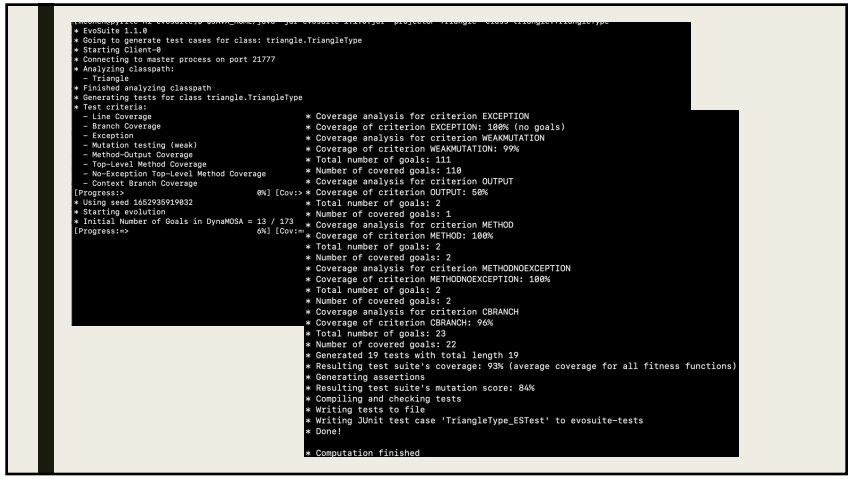
55

Advances in Automated Tools..

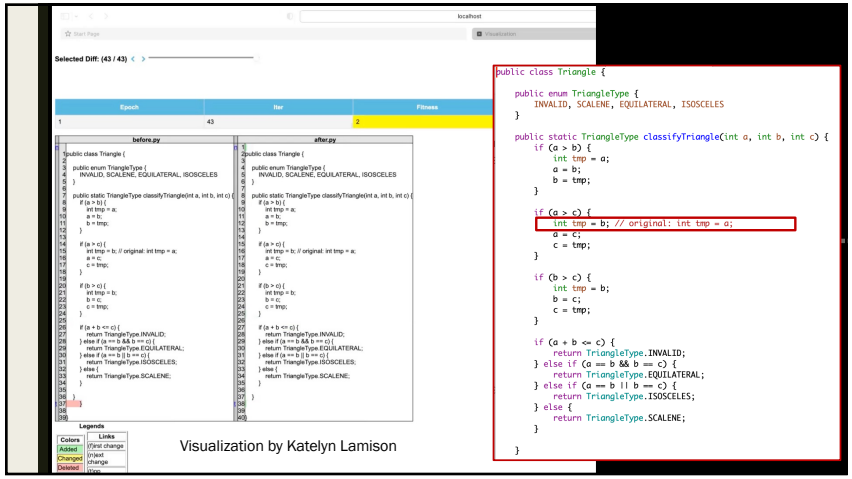
56

```
/**
 * This file was automatically generated by EvoSuite
 * Thu Sep 05 11:10:37 GMT 2019
 */


import org.junit.Test;
import static org.junit.Assert.*;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.junit.runner.RunWith;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = true, useVNET = true, resetStaticState = true, separateClassLoader = true, useJEE = true)
public class TriangleType_ESTest_scaffolding {

  @Test(timeout = 4000)
  public void test00()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(858, 858, 988);
      assertEquals(Triangle.ISOSCELES, triangle0);
  }

  @Test(timeout = 4000)
  public void test01()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(2814, 1968, 2039);
      assertEquals(Triangle.SCALENE, triangle0);
  }

  @Test(timeout = 4000)
  public void test02()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(3, 4, 1);
      assertEquals(Triangle.INVALID, triangle0);
  }

  @Test(timeout = 4000)
  public void test03()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(2, 1, 1);
      assertEquals(Triangle.INVALID, triangle0);
  }

  @Test(timeout = 4000)
  public void test04()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(1, 1, 2);
      assertEquals(Triangle.INVALID, triangle0);
  }

  @Test(timeout = 4000)
  public void test05()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(1692, 1692, 0);
      assertEquals(Triangle.INVALID, triangle0);
  }

  @Test(timeout = 4000)
  public void test06()  throws Throwable  {
      Triangle triangle0 = TriangleType.triangle(2272, 0, (-1));
      assertEquals(Triangle.INVALID, triangle0);
  }
```

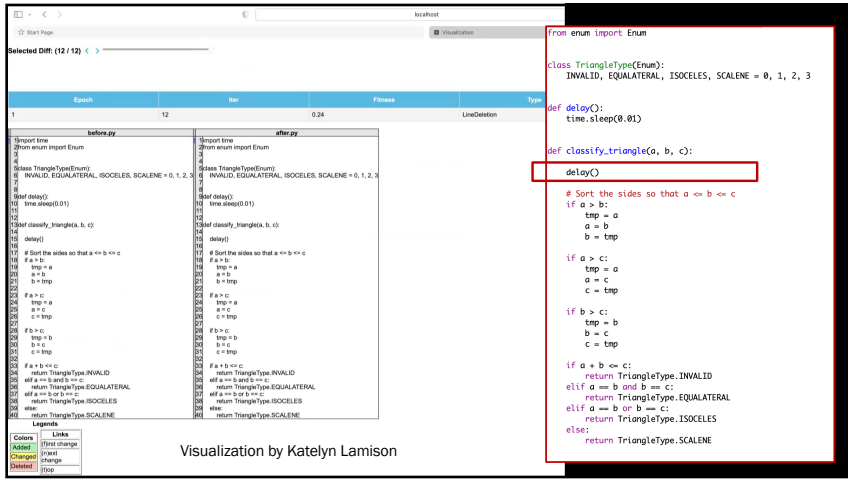
57



58



59



60

Summary

Motivation | What to test | Types of Testing | Models | Coverage | Oracles

61