Programming and Software Development
COMP90041

Lecture 12

Automated Testing & Semester Review

NOTE: Some of the Material in these slides are adopted from
* Lectures Notes prepared by Dr. Peter Schachte, Dr. Rose Williams, and
* the Textbook resources

Udaya Parampalli & Thuan Pham
Semester 1 2021, 24-05-2021

Programming and Software Development
COMP90041

Lecture 12

Automated Testing

NOTE: Some of the Material in these slides are adopted from
* Lectures Notes prepared by Dr. Peter Schachte, Dr. Rose Williams, and
* the Textbook resources

Udaya Parampalli & Thuan Pham
Semester 1 2021, 24-05-2021

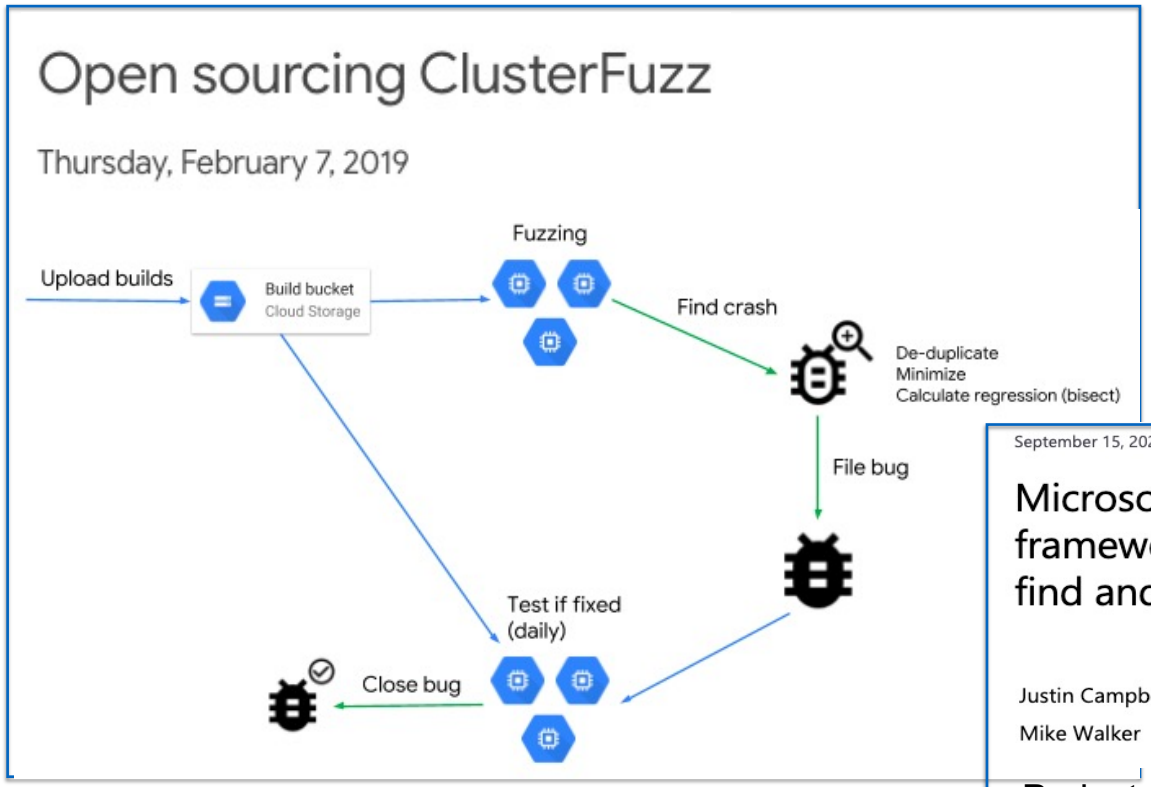
~25,000

In Google products (e.g., Chrome)

~22,500

In 340+ open-source projects integrated with OSS-Fuzz

Bugs found by Fuzzing at Google



September 15, 2020

Microsoft announces new Project OneFuzz framework, an open source developer tool to find and fix bugs at scale

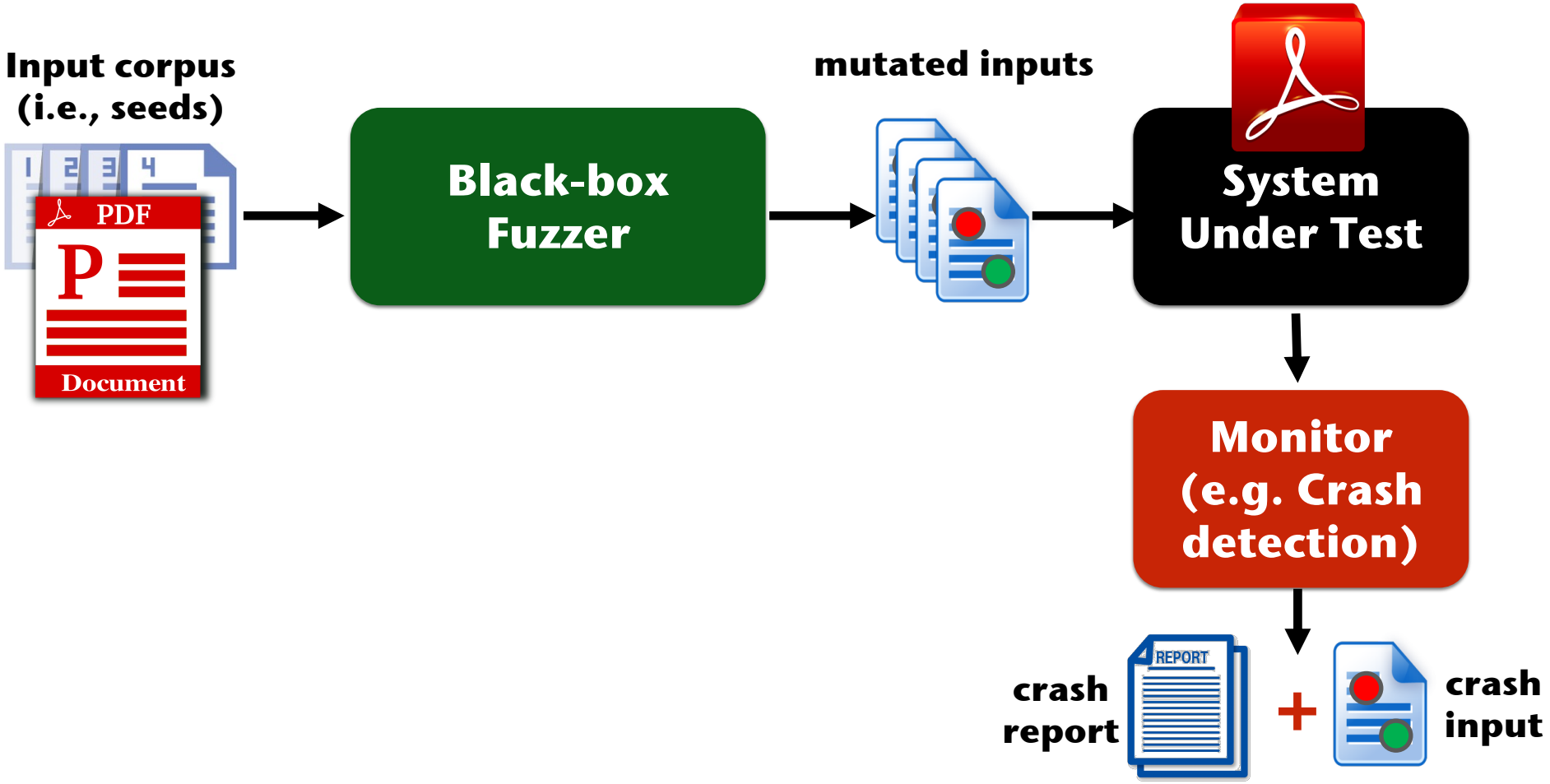
Justin Campbell Principal Security Software Engineering Lead, Microsoft Security

Mike Walker Senior Director, Special Projects Management, Microsoft Security

Project OneFuzz enables continuous developer-driven fuzzing to proactively harden software prior to release. With a single command, which can be baked into CI/CD, developers can launch fuzz jobs from a few virtual machines to thousands of cores.

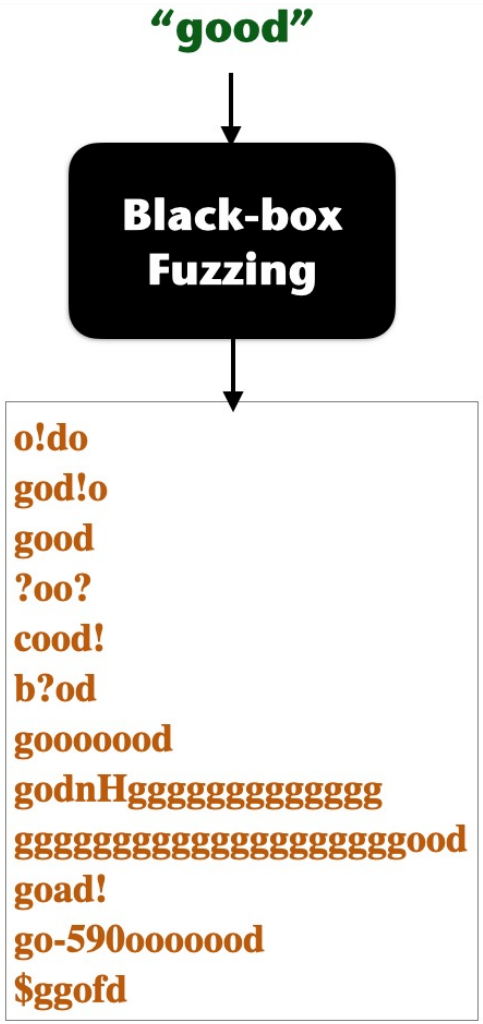
https://github.com/google/clusterfuzz
https://github.com/microsoft/onefuzz

Fuzzing in CI/CD



How does Fuzzing work?

```
void Fn(char buf[4])
{
  if (buf[0] == 'b') {
    if (buf[1] == 'a') {
      if (buf[2] == 'd') {
        if (buf[3] == '!') {
          CRASH();
        }
      }
    }
  }
}
```

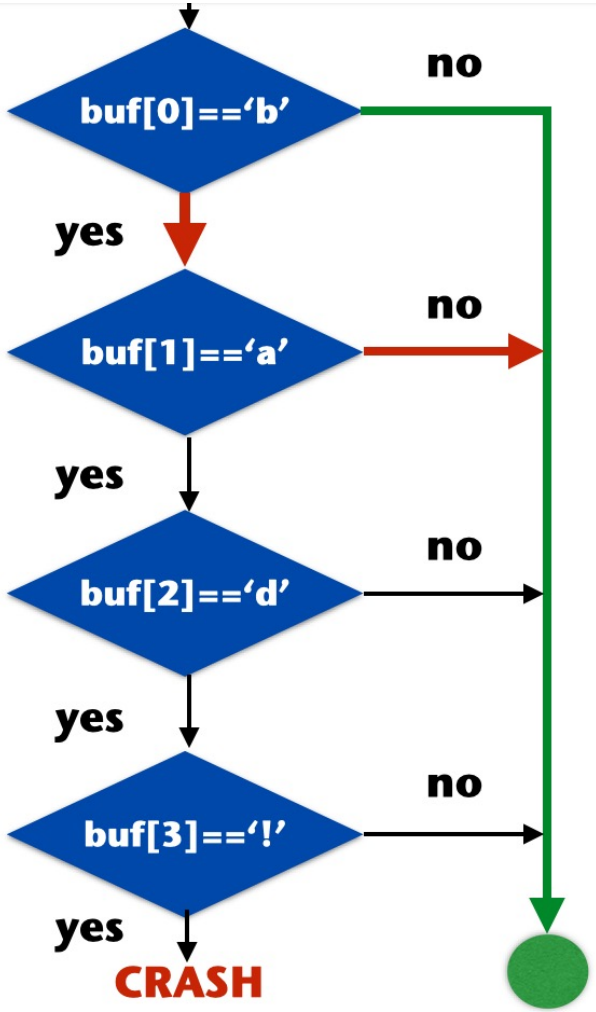


Exampe

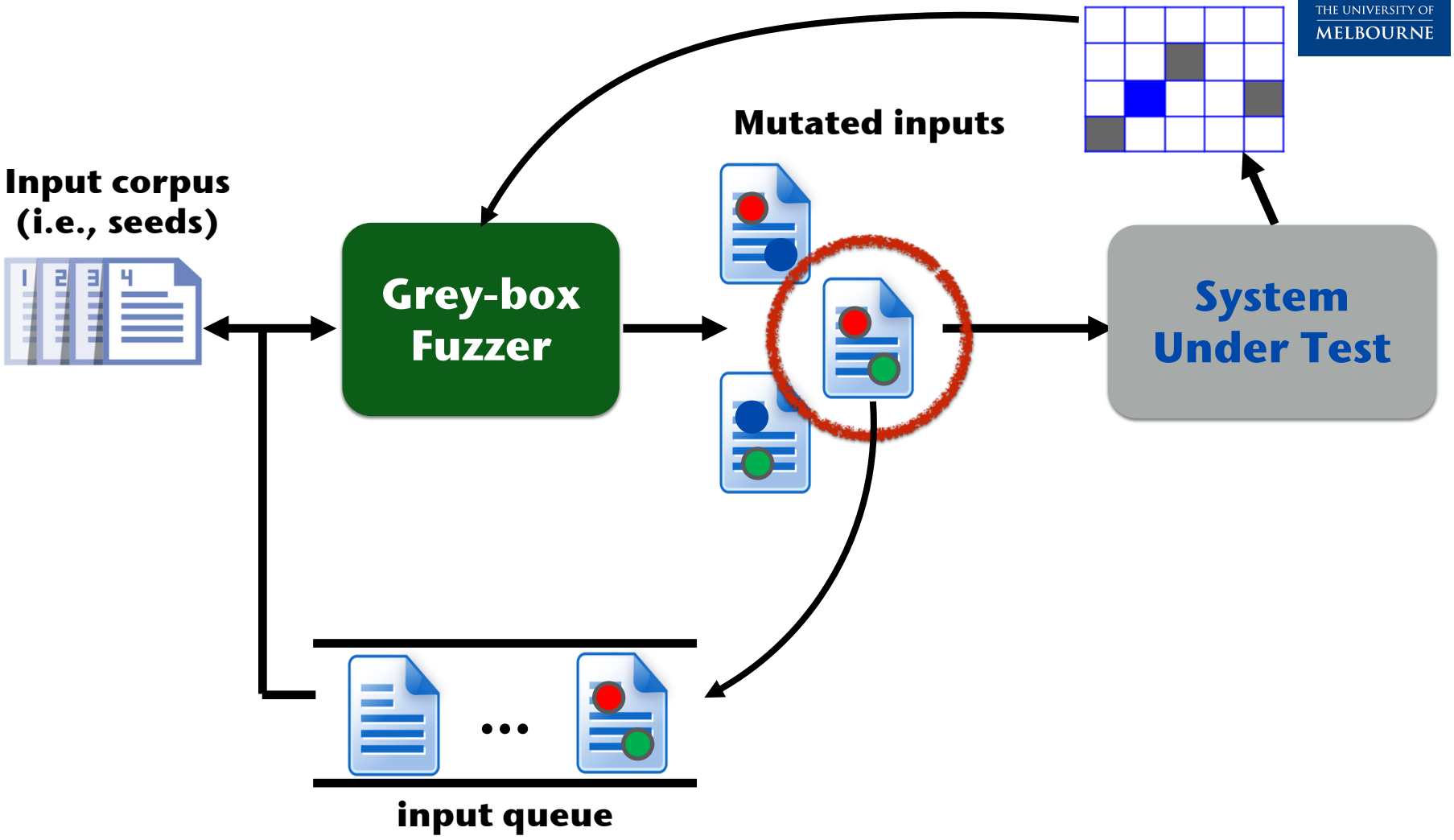
o!do
god!o
good
?oo?
cood!

b?od

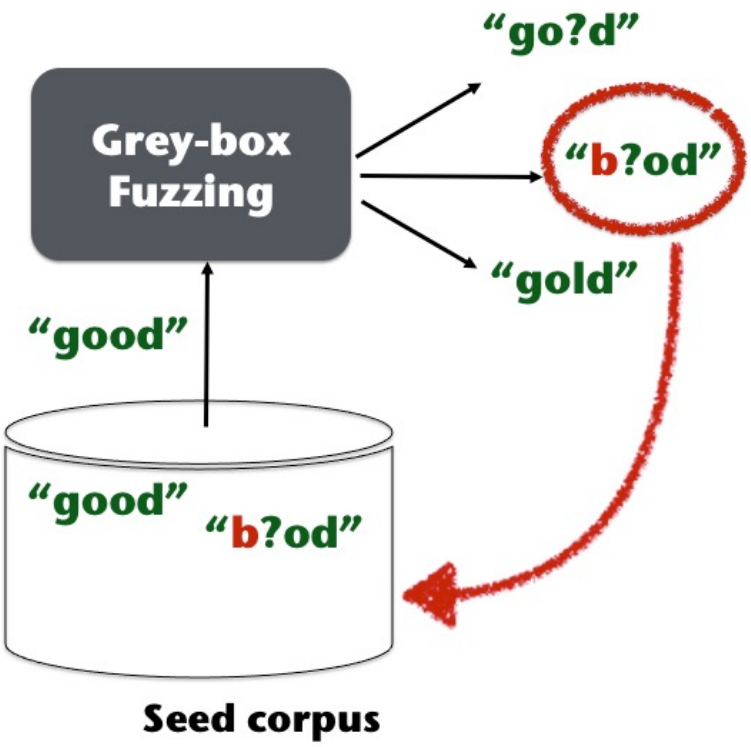
gooooood
godnHgggggggggggg
gggggggggggggggggggood
goad!
go-590ooooood
$ggofd

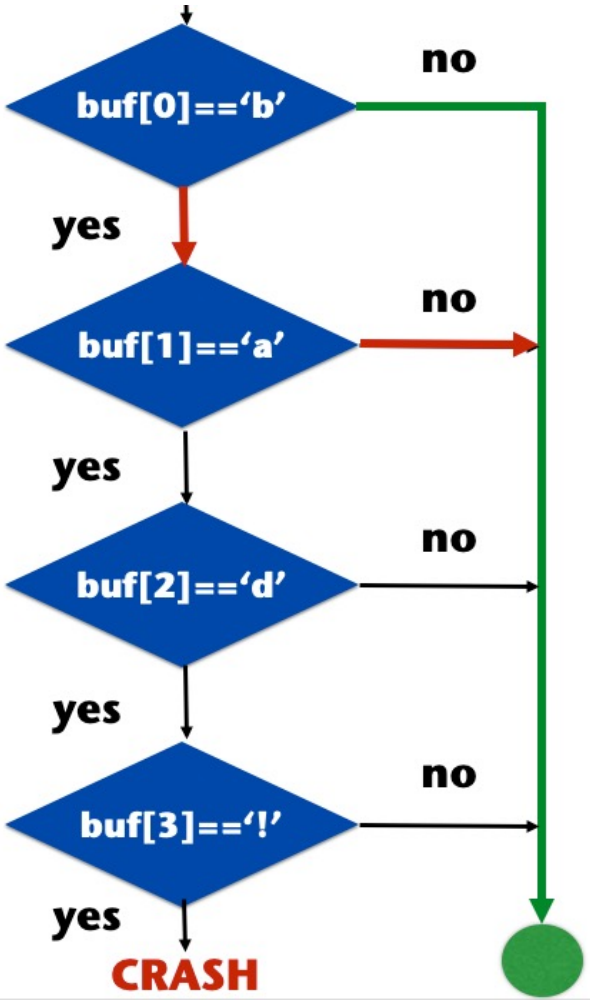


A deeper look

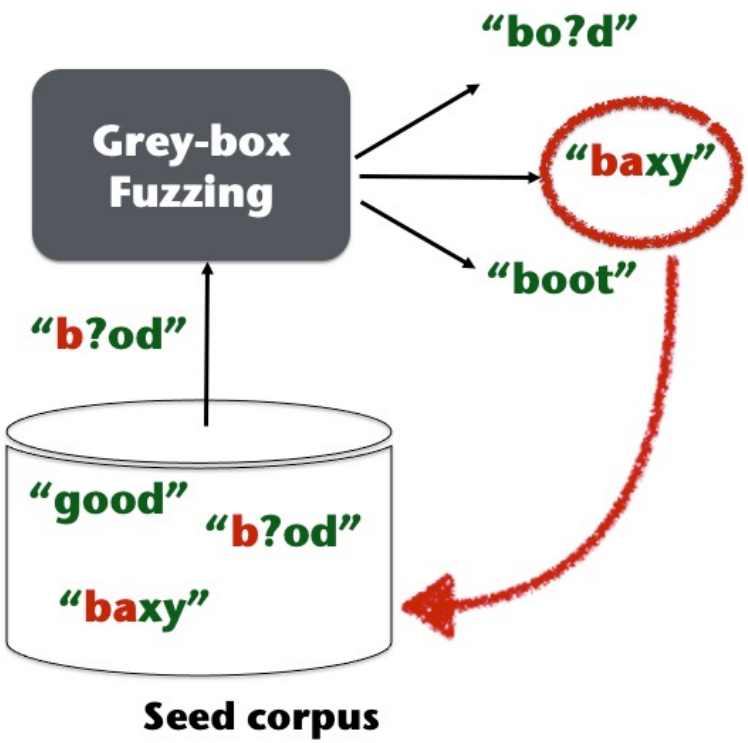


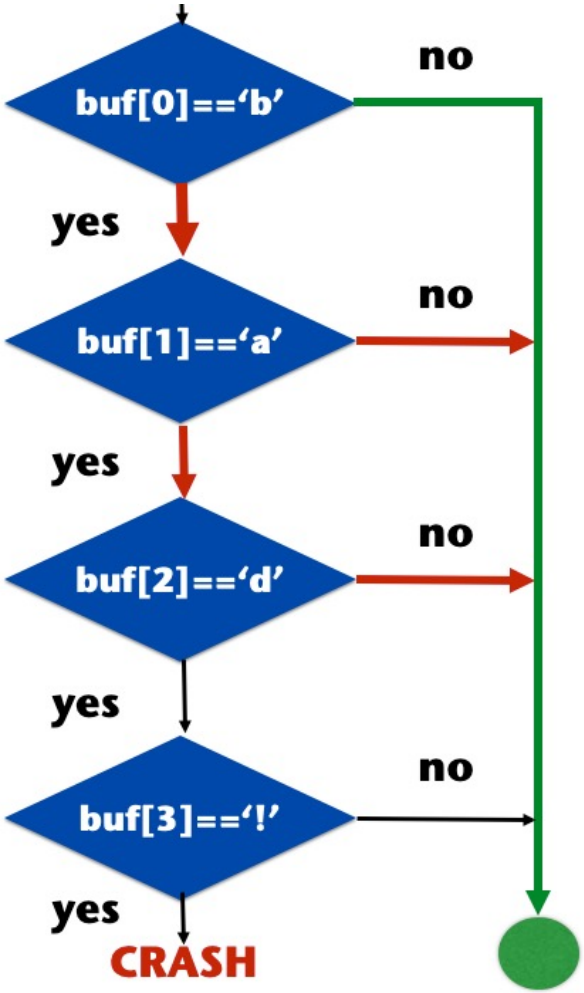
Coverage-Guided Greybox Fuzzing



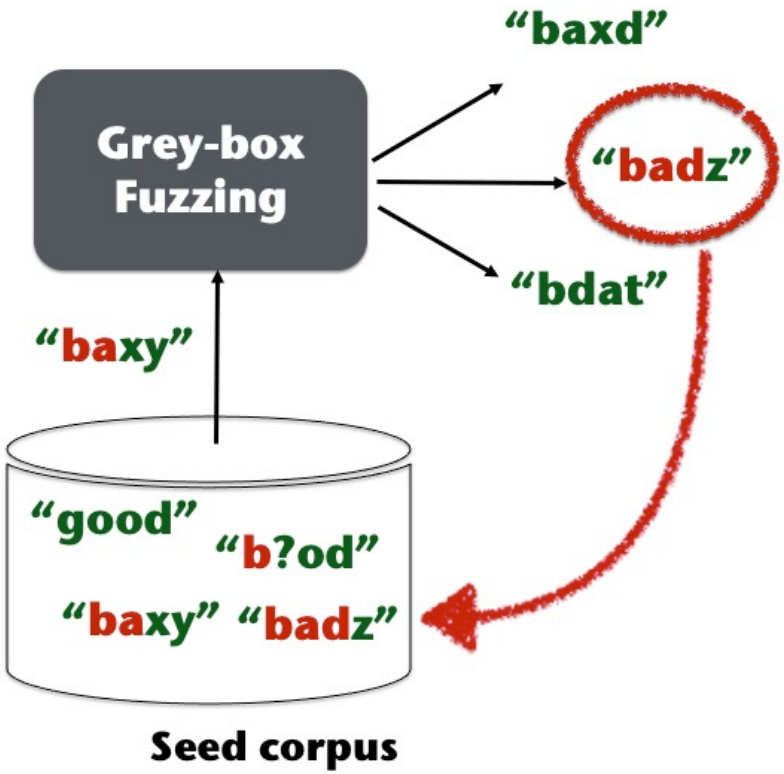


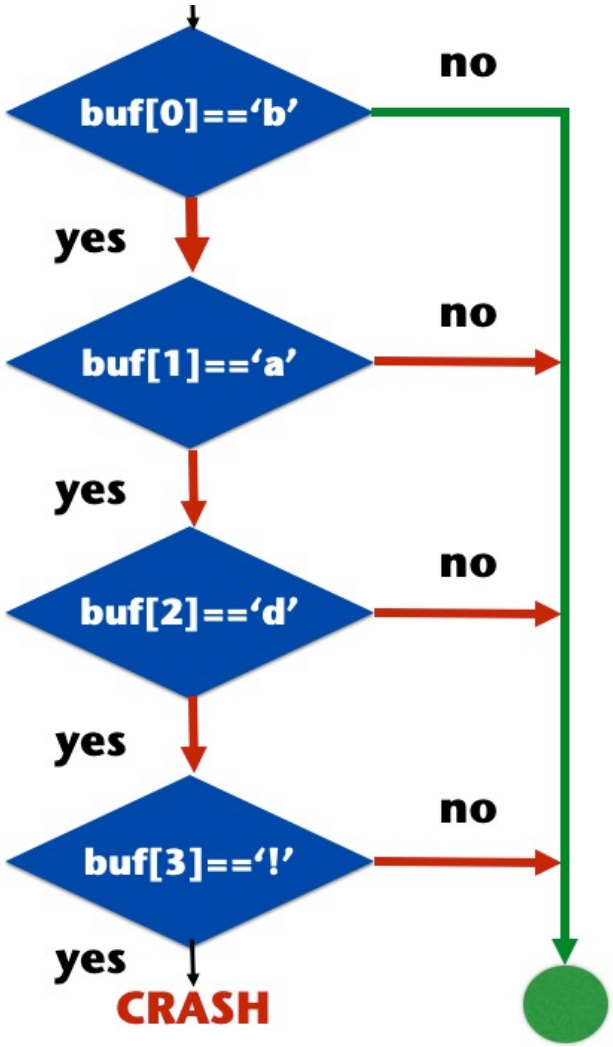
Exampe



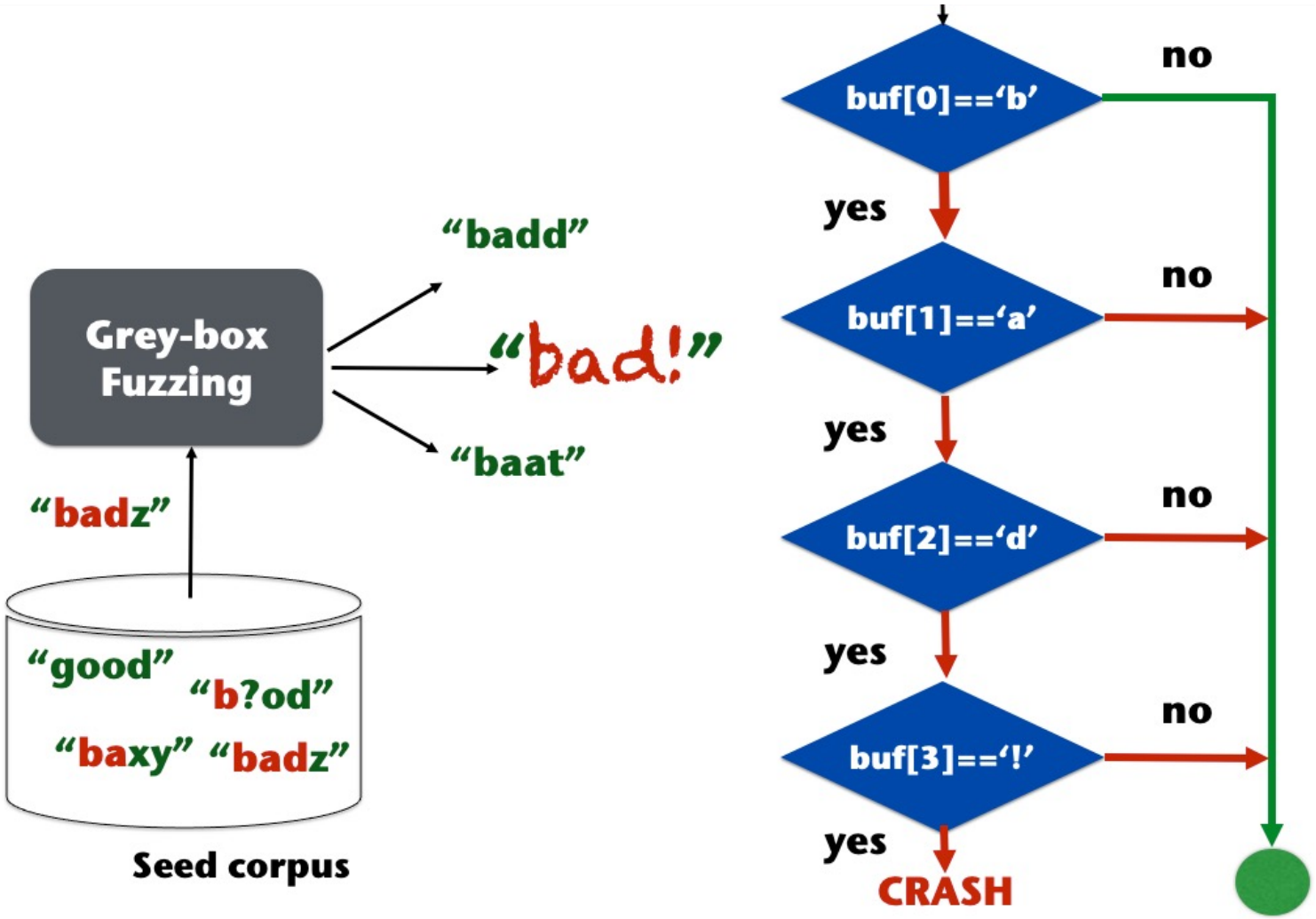


Example (cont'd)





Example (cont'd)



Example (cont'd)

- Black-box fuzzing: JUnit-QuickCheck
  - https://github.com/pholser/junit-quickcheck
- Grey-box fuzzing: JQF
  - JUnit-QuickCheck + Code coverage feedback + Smart algorithms
  - https://github.com/rohanpadhye/JQF

How do we fuzz Java programs?

- JUnit is a ***unit testing*** framework for Java
- JUnit promotes the idea of "first testing then coding" (a.k.a test-driven software development)
- It increases the productivity of programmers and the stability of the software systems
  - Find more bugs and find them ***earlier***
  - Support ***regression testing***

Let's talk about JUnit first

Implement a function `classify` which takes 3 inputs that represent the lengths of the sides of a triangle and returns an integer where the return value

- 1 means it is equilateral (all sides equal length)
- 2 means it is isosceles (exactly 2 equal sides)
- 3 means it is scalene (no equal sides)
- 4 means it is an illegal triangle

```
public int classify(int a, int b, int c)
{
  if (a <= 0 && b <= 0 && c <= 0)
    return 4; //invalid
  if (a <= c - b || b <= a - c || c <= b - a)
    return 4; //invalid
  if (a == b && b == c)
    return 1; //equilateral
  if (a == b || b == c || c == a)
    return 2; //isoscele
  return 3;   //scalene
}
```

Example-How do we test this buggy program?

```java
public static void main(String[] args)
{
  Triangle obj = new Triangle();
  Scanner keyboard = new Scanner(System.in);
  while(true)
  {
    int a, b, c;
    System.out.println("Enter the length of 3 edges:");
    System.out.print("a: ");
    a = keyboard.nextInt();
    System.out.print("b: ");
    b = keyboard.nextInt();
    System.out.print("c: ");
    c = keyboard.nextInt();

    System.out.printf("Given lengths: a=%d, b=%d, c=%d\n", a, b, c);
    int result = obj.classify(a, b, c);

    //print the result ...

    keyboard.nextLine();
    System.out.print("Test more cases? (Y/N): ");
    char selectedOption = Character.toLowerCase(keyboard.next().charAt(0));
    if (selectedOption == 'n') break;
  }
}
```

Triangle.java

Option-1. Manual Testing

- Problems
  - Time consuming & tedious
  - The program should be tested after each (major) changes to
    - Check the correctness of new code
    - Prevent regression bugs
  - Likely to miss ***corner/edge cases***

Option-1. Manual Testing

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class TriangleJUnitTest {

    @Test
    public void testInvalidTriangle() {
        Triangle obj = new Triangle();
        assertEquals(4, obj.classify(-1, 1, 1));
        assertEquals(4, obj.classify(1, 2, 3));
        assertEquals(4, obj.classify(-5, -5, -5));
    }
}
```

TriangleJUnitTest.java

- Bundle test cases in a test class
- The test class can be executed automatically once the code has been updated
- Problem: developers/testers still need to design test inputs

Option-2. Test Automation with JUnit

- Test inputs are generated ***automatically*** in a ***black-box*** *manner*

TriangleQCheckTest.java

```
import java.util.*;
import static org.junit.Assert.*;
import static org.junit.Assume.*;

import org.junit.runner.RunWith;
import com.pholser.junit.quickcheck.Property;
import com.pholser.junit.quickcheck.runner.JUnitQuickcheck;

@RunWith(JUnitQuickcheck.class)
public class TriangleQCheckTest {

    @Property
    public void testInvalidTriangle(int a, int b, int c) {
        assumeTrue(a <= 0 || b <= 0 || c <= 0);
        Triangle obj = new Triangle();
        System.out.printf("\nGenerated lengths: a=%d, b=%d, c=%d\n", a, b, c);
        assertTrue("Invalid triangle", obj.classify(a, b, c) == 4);
    }
}
```

Option-3. Junit-QuickCheck

- Test inputs are generated ***automatically*** in a ***grey-box*** *manner*

TriangleJQFTest.java

```
import java.util.*;
import static org.junit.Assert.*;
import static org.junit.Assume.*;

import org.junit.runner.RunWith;
import com.pholser.junit.quickcheck.*;
import com.pholser.junit.quickcheck.generator.*;
import edu.berkeley.cs.jqf.fuzz.*;

@RunWith(JQF.class)
public class TriangleJQFTest {

    @Fuzz
    public void testInvalidTriangle(int a, int b, int c) {
        assumeTrue(a <= 0 || b <= 0 || c <= 0);
        Triangle obj = new Triangle();
        //System.out.printf("\nGenerated lengths: a=%d, b=%d, c=%d\n", a, b, c);
        assertTrue("Invalid triangle", obj.classify(a, b, c) == 4);
    }
}
```

Option-4. JQF

- Clone the java_test_automation repository
  - git clone https://github.com/thuanpv/java_test_automation.git
- Follow the instructions in README.md
  - Build a Docker image
  - Run a Docker container
  - Compile the source files
  - Run tests
- To learn more about Docker, read this: https://docs.docker.com/get-started/overview/

Demo – Run all examples in a Docker container

- Add more tests e.g., testEquilateralTriangle, testIsosceleTriangle, testScaleneTriangle
- Fuzz these tests using JUnit-QuickCheck and JQF

Homework

Programming and Software Development
COMP90041

Lecture 12

Semester Review

NOTE: Some of the Material in these slides are adopted from
* Lectures Notes prepared by Dr. Peter Schachte, Dr. Rose Williams, and
* the Textbook resources

Udaya Parampalli & Thuan Pham

Semester Review

- Object-Oriented (OO) software development
  - Program design, implementation and testing
  - OO concept
    - classes
    - objects
    - encapsulation
    - inheritance
    - polymorphism
  - The Java programming language
  - Problem solving
    - data structures
    - algorithms

Expectation

- 1: Introduction
  - What a java program looks like? How it works? How to compile and run it, etc
  - Basic operations: primitive types, identifiers, assignment statement, arithmetics, string, etc
- 2: Console Input and Output
  - System.out.println (printf, print), various formats
  - Input using the scanner class, nextInt (nextFloat…), nextLine, etc
- 3: Flow of Control
  - Boolean expressions: logical values, !, &&, ||, >, <, ==, precedence and association rules
  - Branching: if-else; multiway if-else; switch; break; continue
  - Loops: for, while, do-while; nested loop, infinite loop, debugging a loop

Review

- 4: Classes I
  - Type, members (instance variables, methods), local/global variables, this, access permission (public, private), **overloading** (same name different signature), constructors
  - Modularity, information hiding, encapsulation
- 5: Classes II
  - Static methods and variables
  - Modular design
  - References, privacy leak, mutable and immutable classes, equals, toString, packages
  - UML
  - Wrapper classes
- 6: Memory an Arrays
  - Variables in memory
  - Privacy leaks
  - Mutable/immutable types
  - Packages and Javadoc
  - Arrays:
    - Basic operations, references, string array, **No** multidimensional array.
    - Sorting
- 7: Inheritance
  - Base/derived classes, **overriding**, super constructor
  - More access permission
  - Enumerations

Review

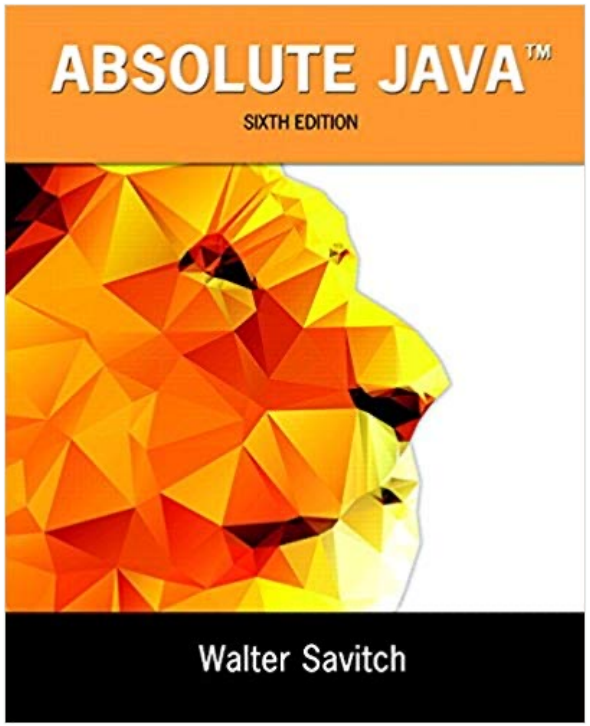
- 8: Polymorphism and Abstract classes
  - Late binding (toString): except static, final, private methods; downcasting/upcasting; **No** clone methods
  - Abstract class: a class containing an abstract method; cannot define an object of an abstract class
  - Interfaces
  - Handling exceptions
    - Try-throw-catch; Exception class; getMessage(); checked/unchecked exceptions
- 9: File I/O
  - Input/Output streams
  - Text file and binary file
  - Textfile: opening, writing (PrintWriter), reading (Scanner, BufferedReader)
  - Buffered Reader
  - Binary Files: ObjectInputStream/ObjectOutputStream, Serializable interface
  - File class (getName, setReadonly, delete, etc)
- 10: Generics & ArrayList
  - ArrayList: basic operations, methods (add, set, get, etc), for-each loop
  - Generics: parameterized classes/methods
  - **No coding required for generic methods**

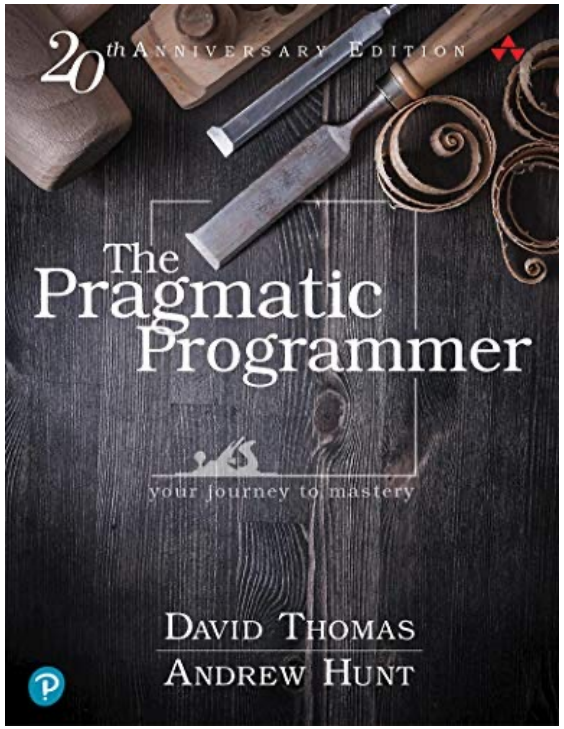
Review

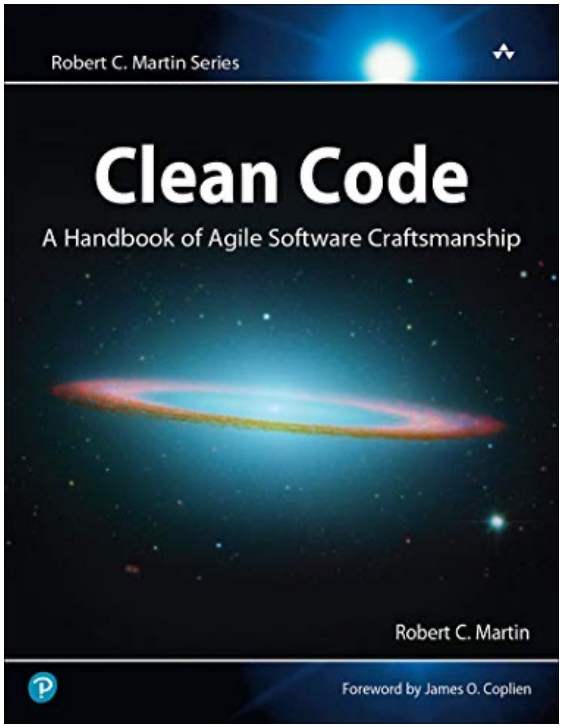
- More advanced subjects
  - Algorithms and complexity
  - Databases
  - Machine learning
  - Security & Software Testing
  - …

- Research projects

What's next?

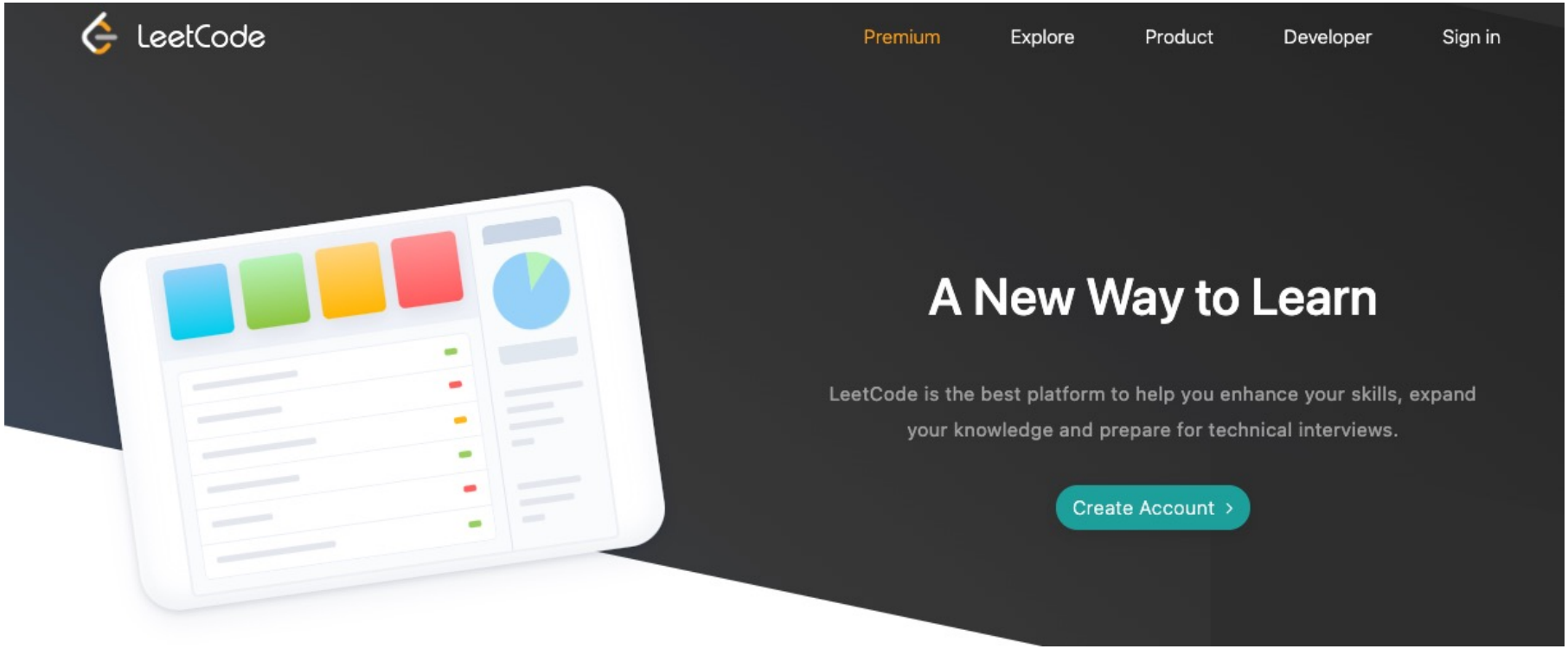






Readings

https://leetcode.com/



And writing more code!

Final Reflections

Thank you all!