AUTOMATED TEST INPUT GENERATION

CS-HU 374 Lecture 8

BASIC ATG APPROACH

ATG tools are called fuzzers

- Generation based
  - Takes a description of valid inputs and generates new inputs based on it
  - Use Java's Context-free grammar to derive valid programs to test compiler implementation
  - Use some formula to generate dependable inputs: generate randomly a and b and then compute $c = \sqrt{a^2 + b^2}$
- Mutation based
  - Starts with existing set of valid inputs, i.e., seeds and then change them in a clever way
  - Changes input values, e.g., (1,-2) becomes (-1,2)
  - Extends sequence of method calls on an object (s.push(a), s.pop()) becomes (s.push(a), s.pop(), s.peek())
- Naive or smart
  - Aware of input structure – grammar, model or protocol based, or other systematic process, e.g., negate values
  - Don't aware – randomly changes inputs. Might find rare inputs but generates many invalid inputs that do not reach program's logic.
- Blackbox, whitebox, or grey-box based
  - Leverage feedback of already executed tests to increase test case diversity
  - Whitebox keeps track of structural elements covered and not covered by previous test cases → Today
  - Blackbox observes results of test cases execution evaluated for further extension → Last lecture

EVOSUITE

EvoSuite – a whitebox fuzzer

Instruments a program to track covered, i.e., executed structural elements of the program

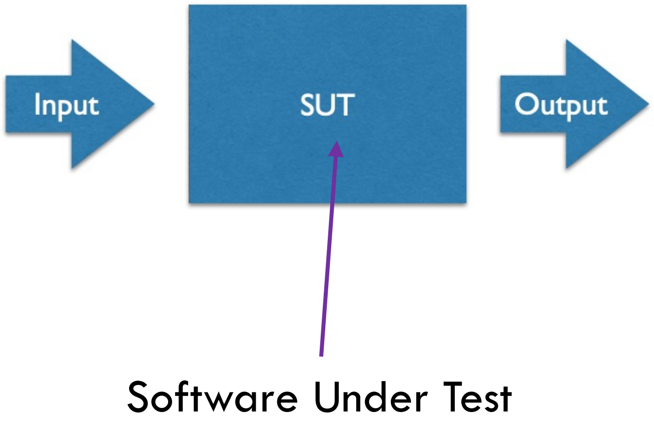
Uses sophisticated genetic search algorithms (from Artificial Intelligence) to generated new inputs

Coverage information a test case is used to evaluate the “fitness” of an input

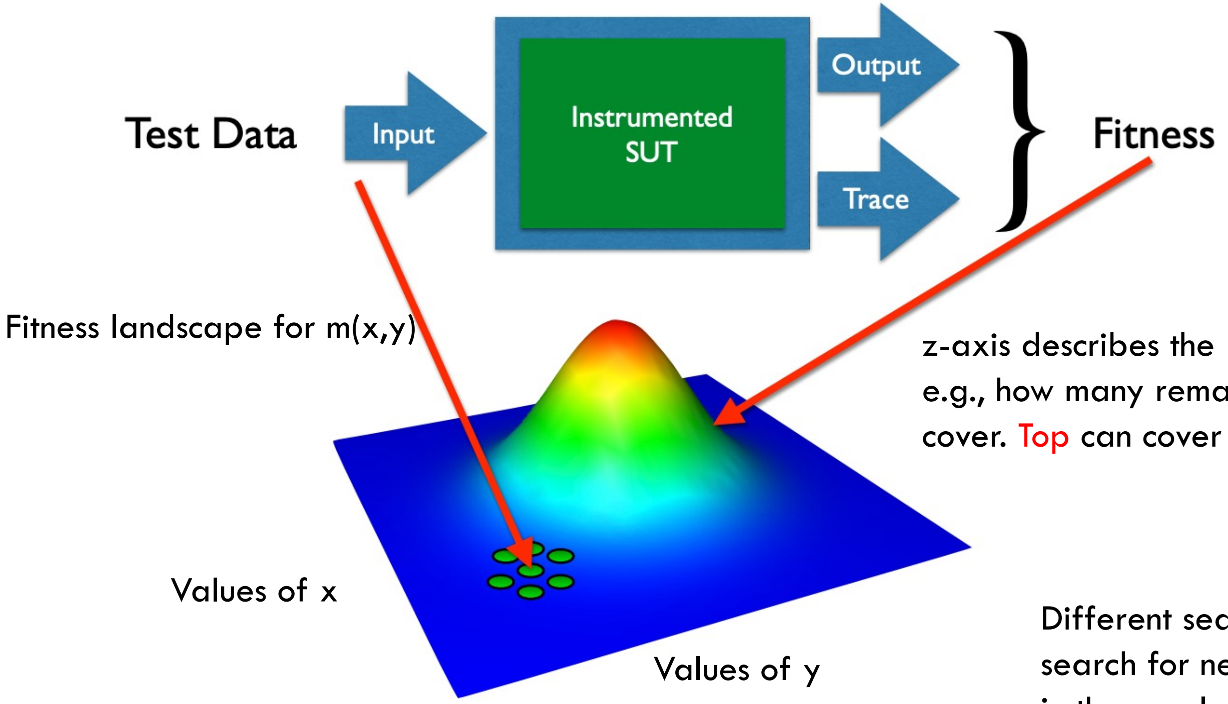
http://www.evosuite.org

SEARCH-BASED TESTING

Regular testing



Search-based testing



z-axis describes the goodness of (x,y) input e.g., how many remaining statements it can cover. Top can cover all statements.

Different search algorithms search for new tests differently in the xy plane.

https://www.st.cs.uni-saarland.de/issta2016/ISSTA16_Tutorial_Fraser.pdf

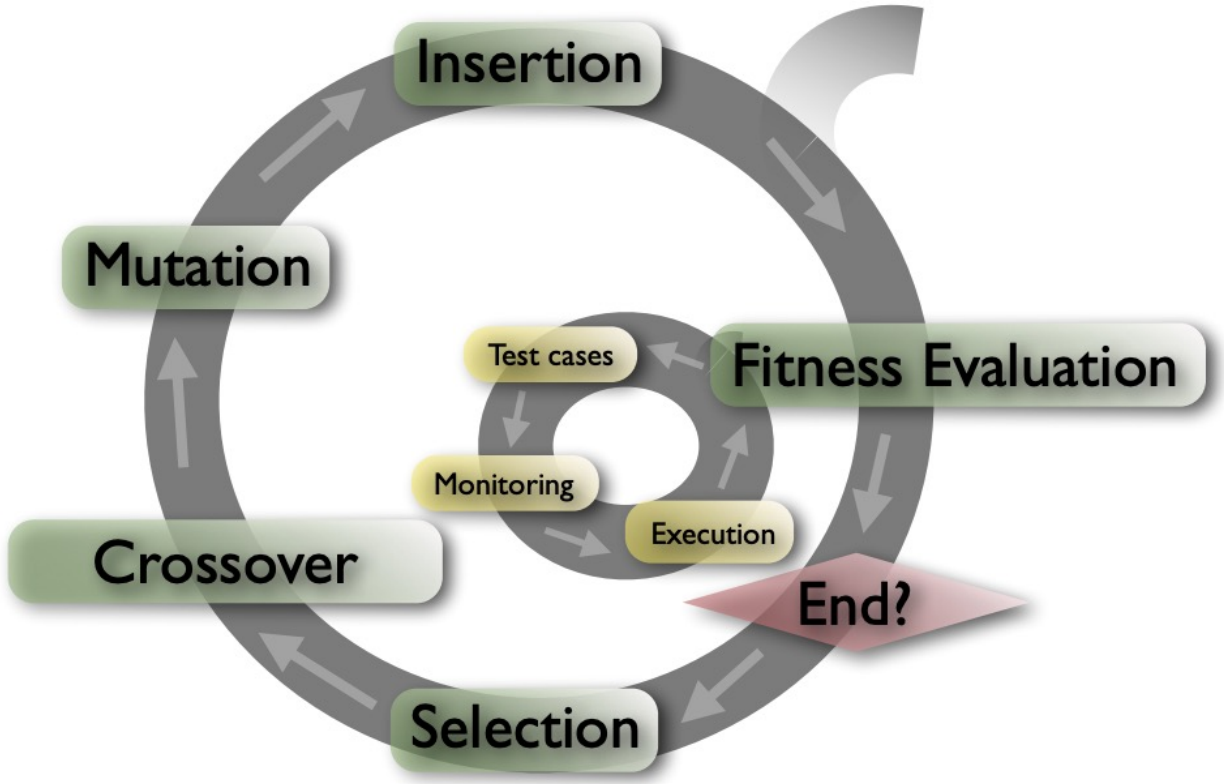
EVOSUITE

Uses evolutionary search algorithm

- Method m(int p1, int p2, int p3)
- Selects input IN1 with a good fitness value, e.g., IN_1 = (-1,0,1) with10% statement coverage
- Selects input IN2 with a good fitness value, e.g., IN_2 = (1,0,-1) with 15% of statement coverage
- Choses where to split inputs, e.g., on the second parameter p2
  - IN_{1_1} = (-1, and IN_{1_2} = 0,1)
  - IN_{2_1} = (1, and IN_{2_2} = 0,-1)
- Creates two new test cases by combining the first and the second parts from different test cases
  - Crossover operation: IN_3 = (-1,0,-1) and IN_4 = (1,0,1)
- Randomly changes, i.e., mutates a single value in some tests
  - $IN_4 \Longrightarrow$ (1,-5,1)
- Produces in a new generation of test cases: IN_3 = (-1,0,-1) and IN_4 = (1,-5,1) (test suite contains now all 4 TCs)
- Evaluates the fitness of new test cases (execute on an instrumented program) and starts the process all over
- Stops when the desired coverage is reached and tries to minimize the resulting test suite

https://www.st.cs.uni-saarland.de/issta2016/ISSTA16_Tutorial_Fraser.pdf

EVOSUITE TESTING OVERVIEW

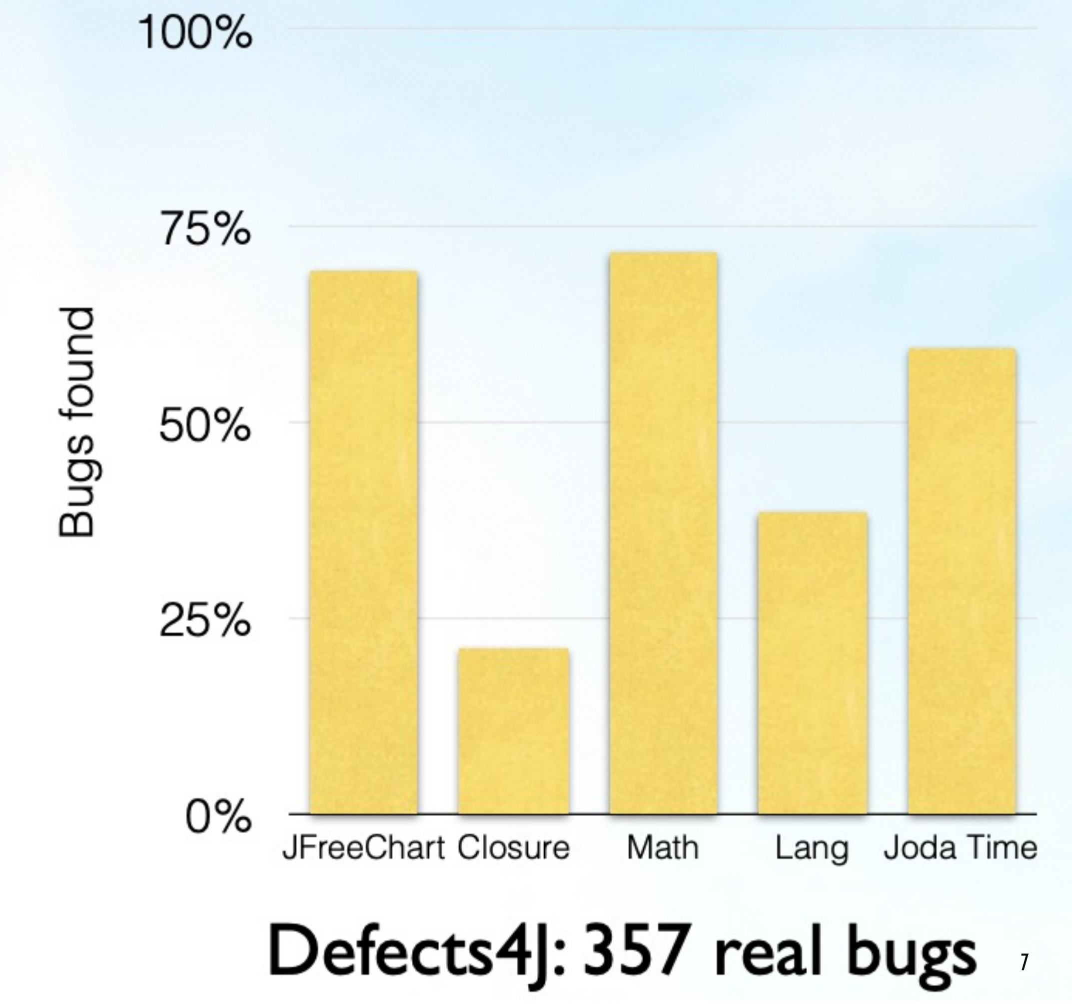


https://www.st.cs.uni-saarland.de/issta2016/ISSTA16_Tutorial_Fraser.pdf

IT WORKS IN REAL WORLD

Several empirical studies

EvoSuite generates test cases that find real bugs



Shamshiri et al. "Do Automatically Generated Unit Tests Find Real Faults? An Empirical Study of Effectiveness and Challenges" ASE, 2015

TO RUN EVOSUITE IN ECLIPSE

It is a command line tool, but we “hack” it to run in Eclipse by calling its main method

Our class under test (CUT) is the TriangleClassifier class in the w4_code package

Also, in w4_code the actual invocation EvoSuiteMain class

```java
package w4_code;

import org.evosuite.EvoSuite;

public class EvoSuiteMain {

    public static void main(String[] args) {
        String[] evoArgs = {"-class", "w4_code.TriangleClassifier", "-projectCP", "./bin/"};
        EvoSuite.main(evoArgs);

    }

}
```

Let’s run it as an application

It runs for some time and create evosuite-tests/code folder with two classes

OUTPUT OF EVOSUITE

It runs for some time and create evosuite-tests/code folder with two classes

Set evosuite-test to be a source folder

Build path → Use as a source folder

EVOSUITE REPORT

It also generates a statistics report

evosuite-report
statistics.csv

It reports search parameters and also code coverage

TARGET_CLASS	criterion	Coverage	Total_Goals	Covered_Goals
code.TriangleClassifier	LINE;BRANCH;EXCEPTION;WEAKMUTATION;OUTPUT;METHOD;METHODNOEXCEPTION;CBRANCH	0.93669355	213	211

You can run TriangleClassifier_EStest.java as a JUnit

But EclEmma gets confused (both tools are instrumentation based)

To run with EclEmma comment out @RunWith line

```
//@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true, useVFS = t
public class TriangleClassifier_ESTest extends TriangleClassifier_ESTest_scaffolding {
```

And run it as JUnit with EclEmma to get the coverage information

Find bugs?

EXPERIMENTING WITH EVOSUITE

Let's try different programs

- Structured programs: w1_code.BoundedStack
- Computational programs: w1_code.OptimizedMultiplier
- With unreachable code: w3_code.Unreachable
- Whole classes: w3_code.Person

In breakout room – ensure that everyone can run the tool for all 4 programs.

Evaluate the coverage statement/branch and number of test cases generated

Compare with your fuzzer, and blackbox tests

NEXT TIME

Working more with EvoSuite

Exploring the list of its tuning parameters

```
String[] help = {"-help"}; /*{"-listParameters"};*/
EvoSuite.main(help);
```

Participation 5 (last one) on tuning EvoSuite

Assigning homework 3 (last one) on tuning EvoSuite