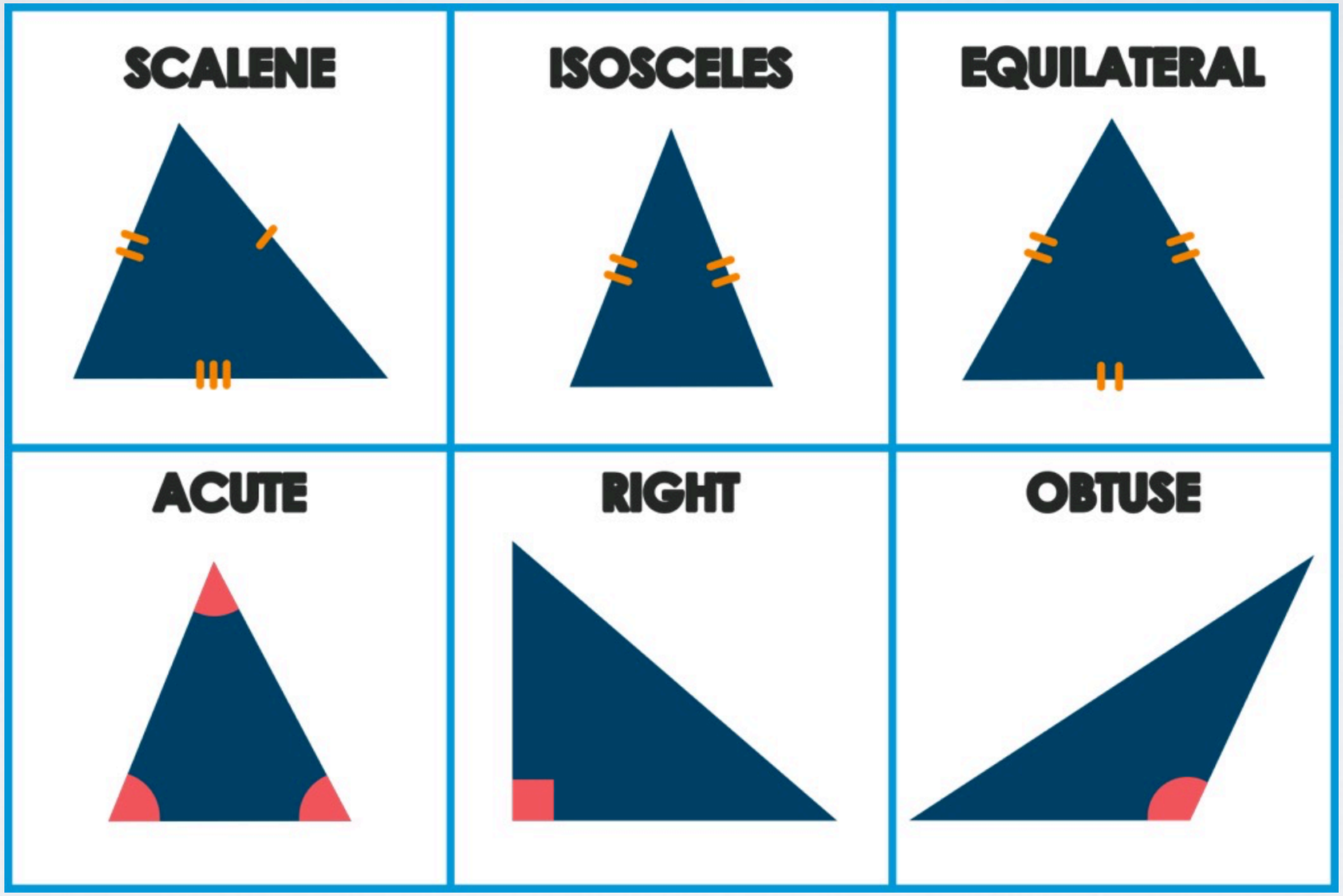
Search-based Software Testing

Annibale Panichella
a.panichella@tudelft.nl

Outline

- White-box Unit Testing
- Random Testing (Fuzzing)
- Search-based Software Testing
- Genetic Algorithms
- Test Case Generation
- Tools

Triangle Shapes



Unit Testing

Class Under Test (CUT)

```
class Triangle {
 int a, b, c; //sides
 String type = "NOT_TRIANGLE";

 Triangle (int a, int b, int c){…}

 void computeTriangleType() {
  1.  if (a == b) {
  2.      if (b == c)
  3.          type = "EQUILATERAL";
          else
  4.          type = "ISOSCELES";
      } else {
  5.      if (a == c) {
  6.          type = "ISOSCELES";
          } else {
  7.          if (b == c)
  8.              type = “ISOSCELES”;
              else
  9.              type = “SCALENE”;
          }
      }
}
```

Unit Testing

Class Under Test (CUT)

```
class Triangle {
 int a, b, c; //sides
 String type = "NOT_TRIANGLE";

 Triangle (int a, int b, int c){…}

 void computeTriangleType() {
  1.  if (a == b) {
  2.      if (b == c)
  3.          type = "EQUILATERAL";
          else
  4.          type = "ISOSCELES";
      } else {
  5.      if (a == c) {
  6.            type = "ISOSCELES";
         } else {
  7.          if (b == c)
  8.                type = “ISOSCELES”;
              else
  9.              type = “SCALENE”;
         }
      }
}
```

Test Case

```
@Test
public void test(){
   // Constructor (init)
   // Method Calls
   // Assertions  (check)
}
```

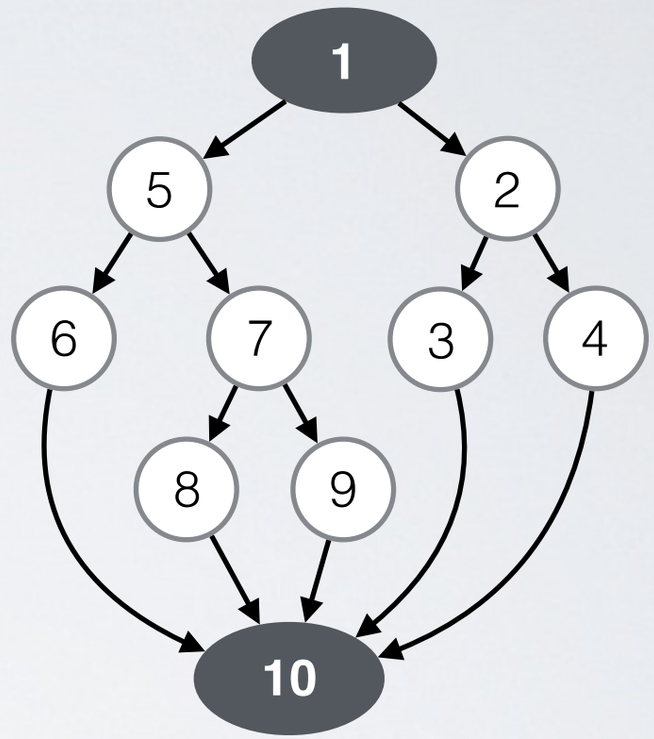
```
@Test
public void test(){
 Triangle t = new Triangle (1,2,3);
 t.computeTriangleType();
 String type = t.getType();
 assertTrue(type.equals(“SCALENE”));
}
```

Unit Testing

Class Under Test (CUT)

```
class Triangle {
 int a, b, c; //sides
 String type = "NOT_TRIANGLE";

 Triangle (int a, int b, int c){…}

 void computeTriangleType() {
  1.  if (a == b) {
  2.      if (b == c)
  3.          type = "EQUILATERAL";
          else
  4.          type = "ISOSCELES";
      } else {
  5.      if (a == c) {
  6.          type = "ISOSCELES";
          } else {
  7.          if (b == c)
  8.                  type = “ISOSCELES”;
                else
  9.                type = “SCALENE”;
          }
      }
}
```

Control Flow Graph

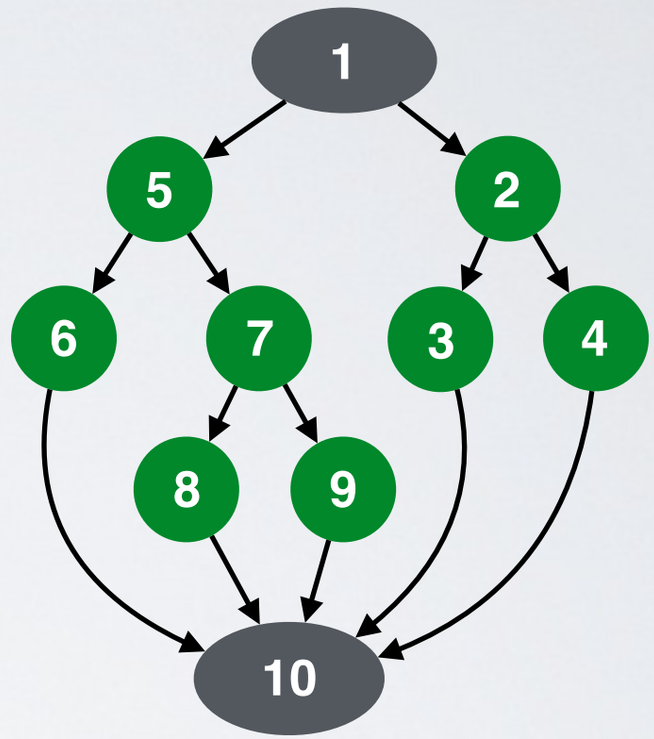


Unit Testing

Class Under Test (CUT)

```
class Triangle {
 int a, b, c; //sides
 String type = "NOT_TRIANGLE";

 Triangle (int a, int b, int c){…}

 void computeTriangleType() {
  1.  if (a == b) {
  2.      if (b == c)
  3.          type = "EQUILATERAL";
          else
  4.          type = "ISOSCELES";
      } else {
  5.      if (a == c) {
  6.          type = "ISOSCELES";
          } else {
  7.          if (b == c)
  8.                  type = “ISOSCELES”;
                else
  9.                type = “SCALENE”;
          }
      }
}
```

Control Flow Graph



Goal: Covering as many code elements as possible

How Much to Test?

Statement coverage

Targets = {1, 2, 3,4 ,5 ,6, 7, 8, 9,10}

Branch coverage

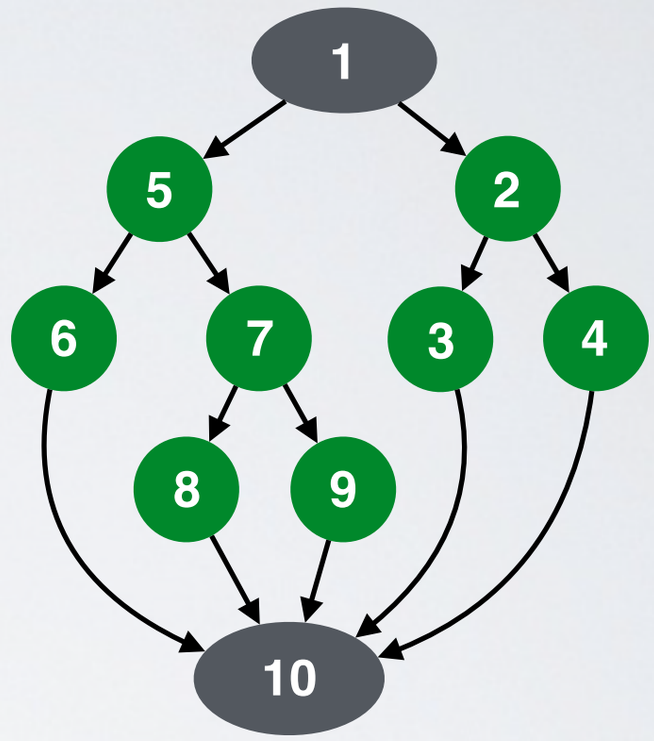
Targets = {<1,5>, <1,2>,<5,6>, <5,7>,
<2,3>, <2,4>, <6,10>, <7,8>, <7,9>,
<3,10>, <4,10>, <8,10>, <9,10>}

Path coverage

Targets = {<1,5,6,10>, <1,5,7,8,10>,
<1,5,7,9,10>, <1,2,3,10>, <1,2,4,10>}

Mutation Coverage?

Control Flow Graph



Why SBSE in Unit Testing?

TU Delft

Project = Apache commons BCEL
Class = Pass2Verifier.java

```
1  /*
2   * Licensed to the Apache Software Foundation (ASF) under one or more
3   * contributor license agreements.  See the NOTICE file distributed with
4   * this work for additional information regarding copyright ownership.
5   * The ASF licenses this file to You under the Apache License, Version 2.0
6   * (the "License"); you may not use this file except in compliance with
7   * the License.  You may obtain a copy of the License at
8   *
9   *      http://www.apache.org/licenses/LICENSE-2.0
10  *
11  *  Unless required by applicable law or agreed to in writing, software
12  *  distributed under the License is distributed on an "AS IS" BASIS,
13  *  WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14  *  See the License for the specific language governing permissions and
15  *  limitations under the License.
16  *
17  */
18 package org.apache.bcel.verifier.statics;
19
20
21 import java.util.HashMap;
22 import java.util.HashSet;
23 import java.util.Locale;
24 import java.util.Map;
25 import java.util.Set;
26
27 import org.apache.bcel.Const;
28 import org.apache.bcel.Constants;
29 import org.apache.bcel.Repository;
30 import org.apache.bcel.classfile.Attribute;
31 import org.apache.bcel.classfile.ClassFormatException;
32 import org.apache.bcel.classfile.Code;
33 import org.apache.bcel.classfile.CodeException;
34 import org.apache.bcel.classfile.Constant;
35 import org.apache.bcel.classfile.ConstantClass;
36 import org.apache.bcel.classfile.ConstantDouble;
```

Random Testing (Fuzzing)

Random Testing

TU Delft

Class Under Test API

```
public ClassUT(…)
…
public ClassUT(…)

public method1(…)
…
public methodK(…)
```

Random Testing

TU Delft

Class Under Test API

```
public ClassUT(…)
…
public ClassUT(…)
```

Constructors

```
public method1(…)
…
public methodK(…)
```

Public Methods

Random Test

```
@Test
public void test(){
  // constructor
  // method calls
  // assertions
}
```

Random Testing

TU Delft

Class Under Test API

```
public ClassUT(…)
…
public ClassUT(…)

public method1(…)
…
public methodK(…)
```

Pick one of the available constructors

With random parameters input

Random Test

```
@Test
public void test(){
  ClassUT c = new ClassUT(.);
  // method calls
  // assertions
}
```

Random Testing

Class Under Test API

```
public ClassUT(…)
…
public ClassUT(…)

public method1(…)
…
public methodK(…)
```

Pick one or more public methods

With random parameters input

Random Test

```
@Test
public void test(){
  ClassUT c = new ClassUT(.);
  c.method1(…);
  c.method3(…);
  // assertions
}
```

Random Testing

TU Delft

Class Under Test API

```
public ClassUT(…)
…
public ClassUT(…)

public method1(…)
…
public methodK(…)
```

Use get methods to check the state of the objects after execution

Random Test

```
@Test
public void test(){
  ClassUT c = new ClassUT(.);
  c.method1(…);
  c.method3(…);
  assertTrue(me.method2());
}
```

Random Testing

TU Delft

How to generate random tests:

1) Pick one of the available constructors (with random input)

2) Pick one or more public methods (with random input)

3) Generate the assertions by checking the final state of the object using get methods

Running Example

```
class PeakFunction {
   private double x, y;

   public PeakFunction (double pX, double pY){
      x = pX; y = pY;
   }

   public double computeZeta() {
     double z =  3*Math.pow(1-x,2);
     z *= Math.exp(- Math.pow(x,2) - Math.pow(y+1,2));
     z -= 10*(x/5 - Math.pow(x,3);
     z -= Math.pow(y,5)*Math.exp(-Math.pow(x,2)-Math.pow(x,2));
     z -= 1/3* Math.exp(-Math.pow(x+1,2) - Math.pow(y,2));
     return z;
   }

   public boolean isZero() {
   if (computeZeta() <= 0.05)
      return true;
   }
}
```

We want to cover this method

Running Example

TU Delft

PeakFunction API

```
// Public constructors
PeakFunction(double, double)

// Public methods
public double computeZeta()
public boolean isZero()
```

Random Test

```
@Test
public void test(){
    // constructor
    // method calls
    // assertions
}
```

Running Example

TU Delft

PeakFunction API

```
// Public constructors
PeakFunction(double, double)

// Public methods
public double computeZeta()
public boolean isZero()
```

Random Test

```
@Test
public void test(){
    PeakFunction pf = new PeakFunction(?,?);
    boolean b = isZero();
    assertTrue(b); or assertFalse(b);
}
```

To generate random double in [0;1], we can use the method **Math.random()**

To generate random double in [a;b], we can use the following code:
Math.random() * (a+b) - a

Running Example

TU Delft

PeakFunction API

```
// Public constructors
PeakFunction(double, double)

// Public methods
public double computeZeta()
public boolean isZero()
```

```
x = 0.8147 , y = 0.9058, z = 2.2346
x = 0.1270 , y = 0.9134, z = 2.6334
x = 0.6324 , y = 0.0975, z = 0.8979
x = 0.2785 , y = 0.5469, z = 0.1855
```

Random Test1

```
@Test
public void test(){
    PeakFunction pf = new PeakFunction(1.4, 2.3);
    boolean b = isZero(); // zeta = 0.4746
    assertFalse(b);
}
```

Random Test2

```
@Test
public void test(){
    PeakFunction pf = new PeakFunction(-0.2, 1.5);
    boolean b = isZero(); // zeta =7.7118
    assertFalse(b);
}
```

Running Example

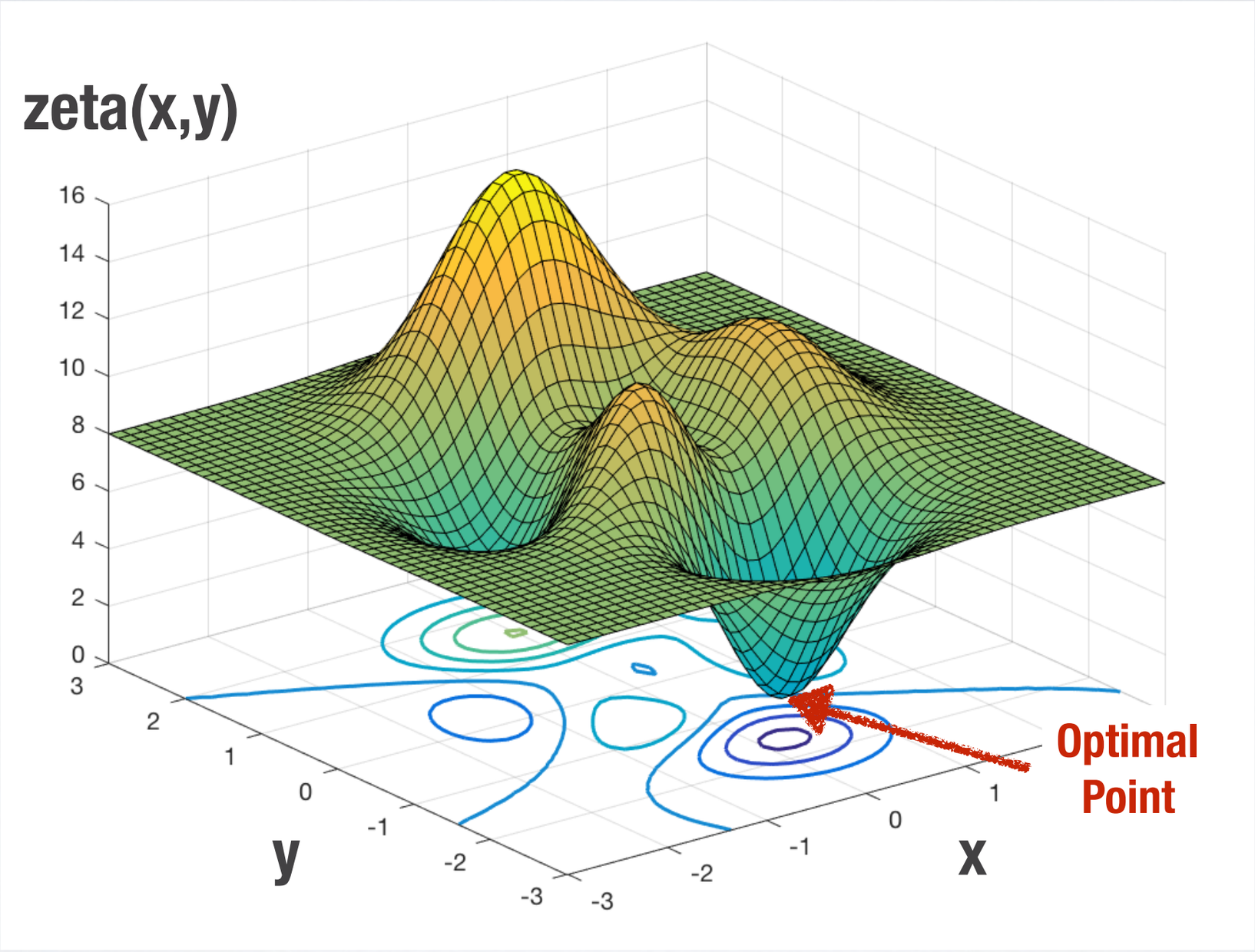
```
class PeakFunction {
    ...
    public boolean isZero() {
    if (computeZeta() <= 0.05)
        return true;
    }
}
```

The false branch is very easy to cover with random testing

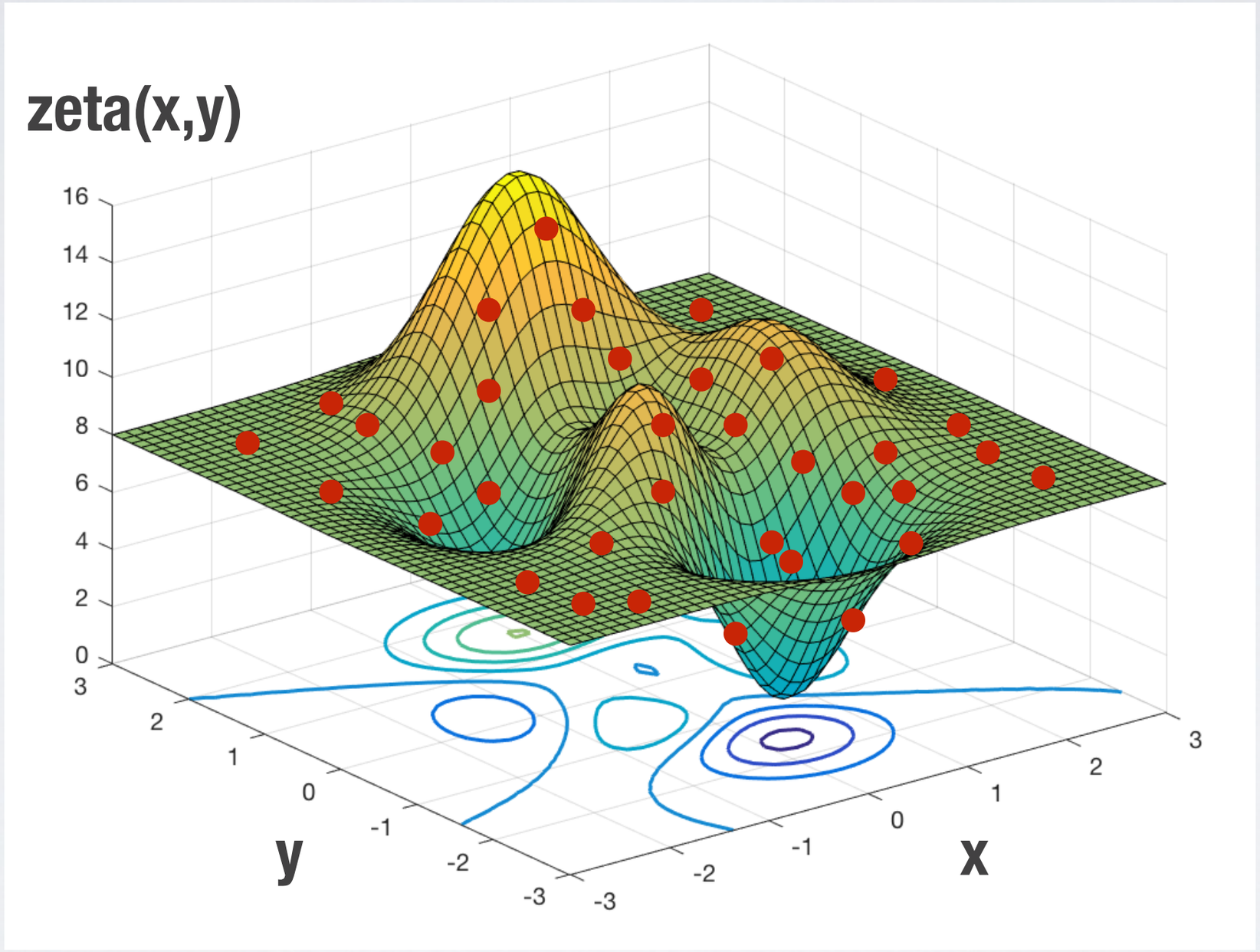
The true branch is very difficult as it require to satisfy the equation computeZeta()=0

What is the probability that a random pair of number X and Y satisfy that condition?

Running Example



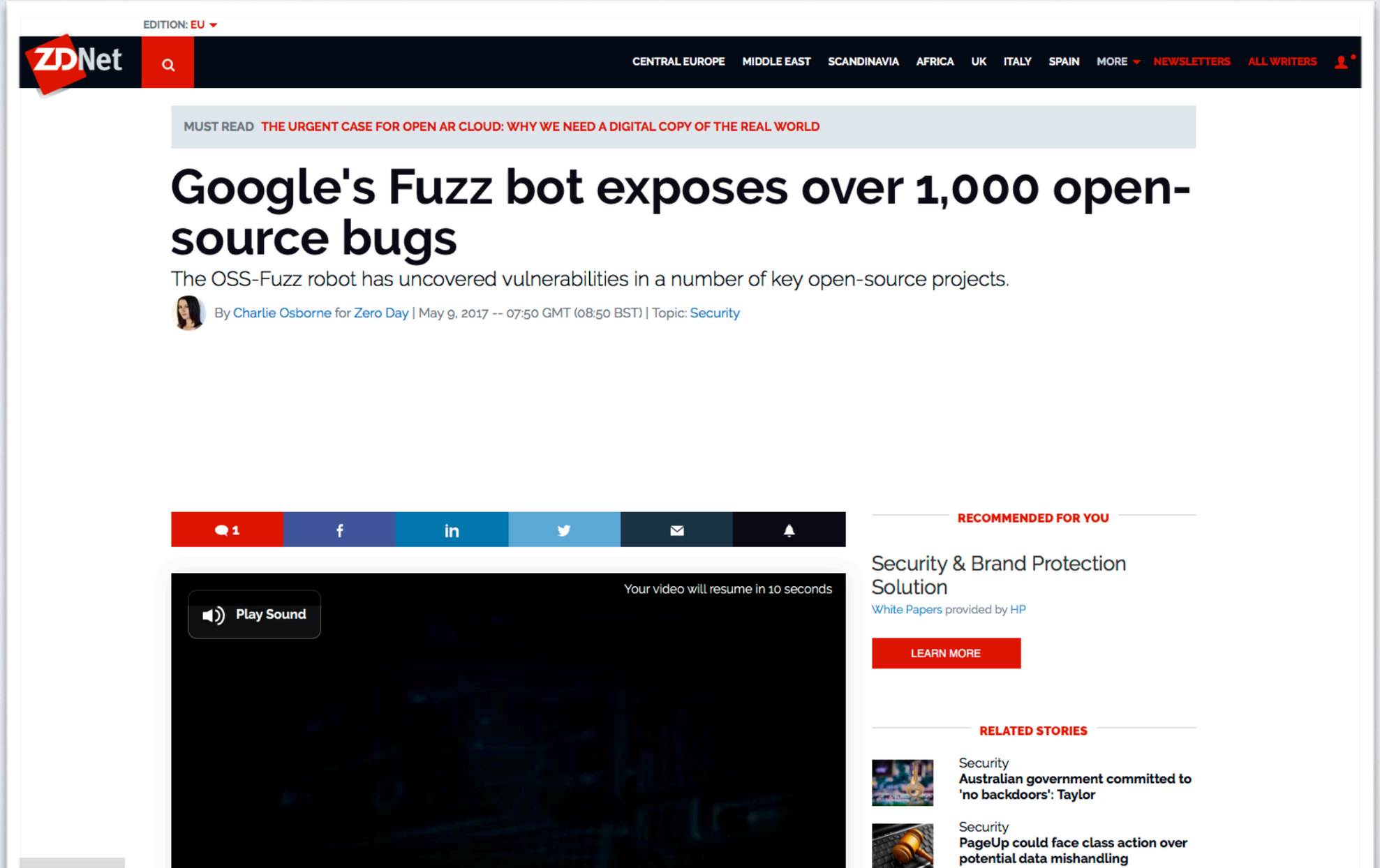
Running Example



Random Testing in Practice

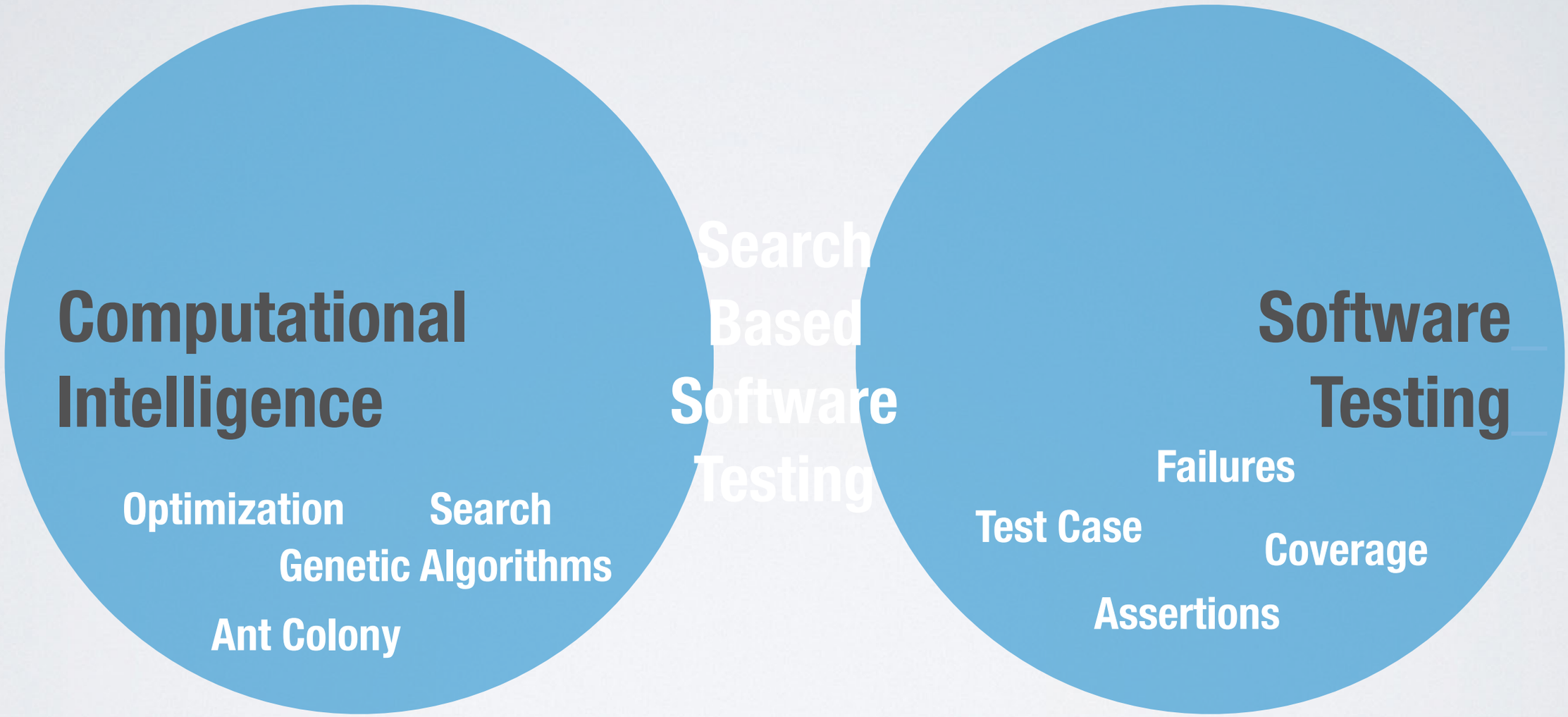
TU Delft

https://www.owasp.org/index.php/Fuzzing

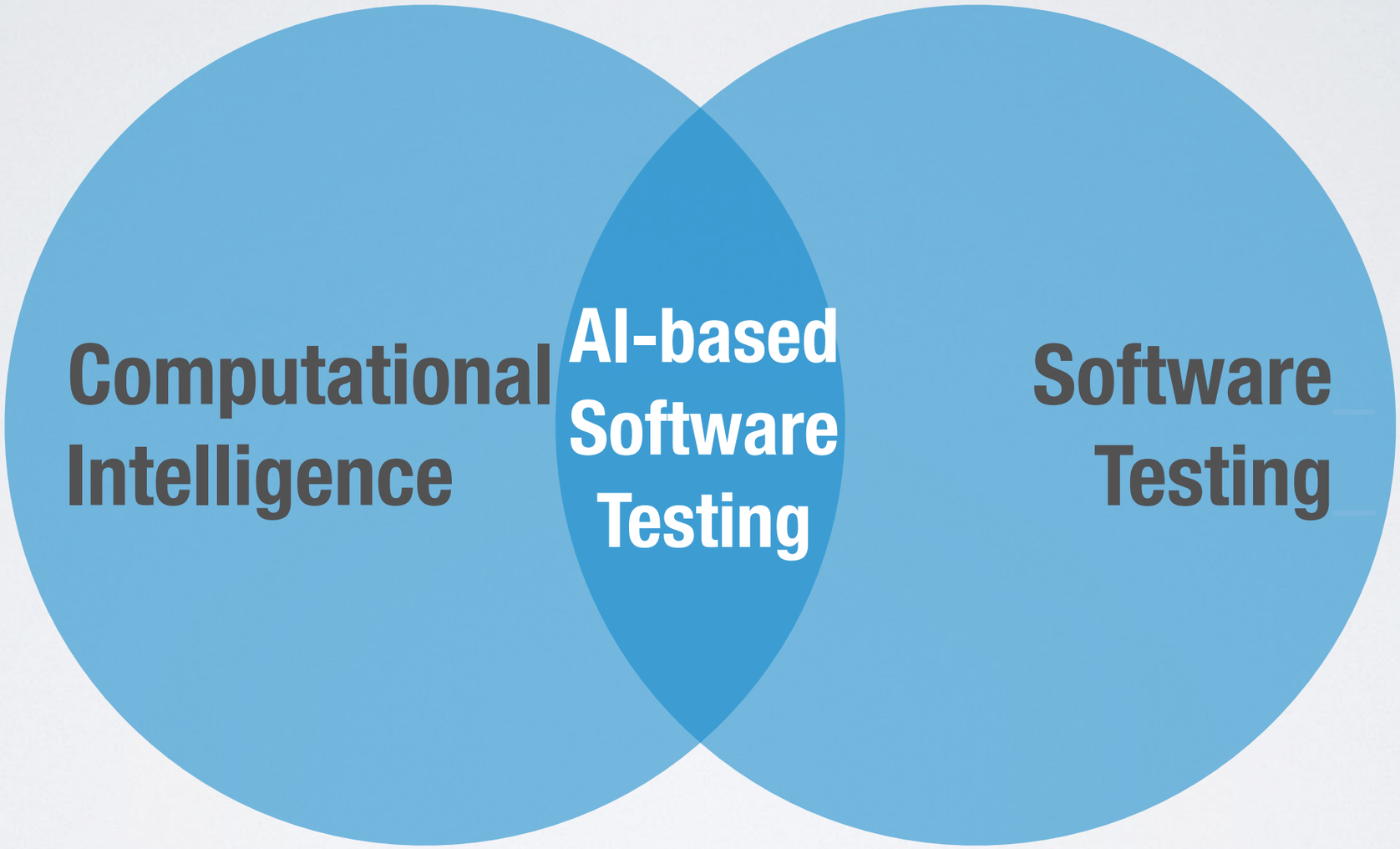


Search-Based Test Case Generation

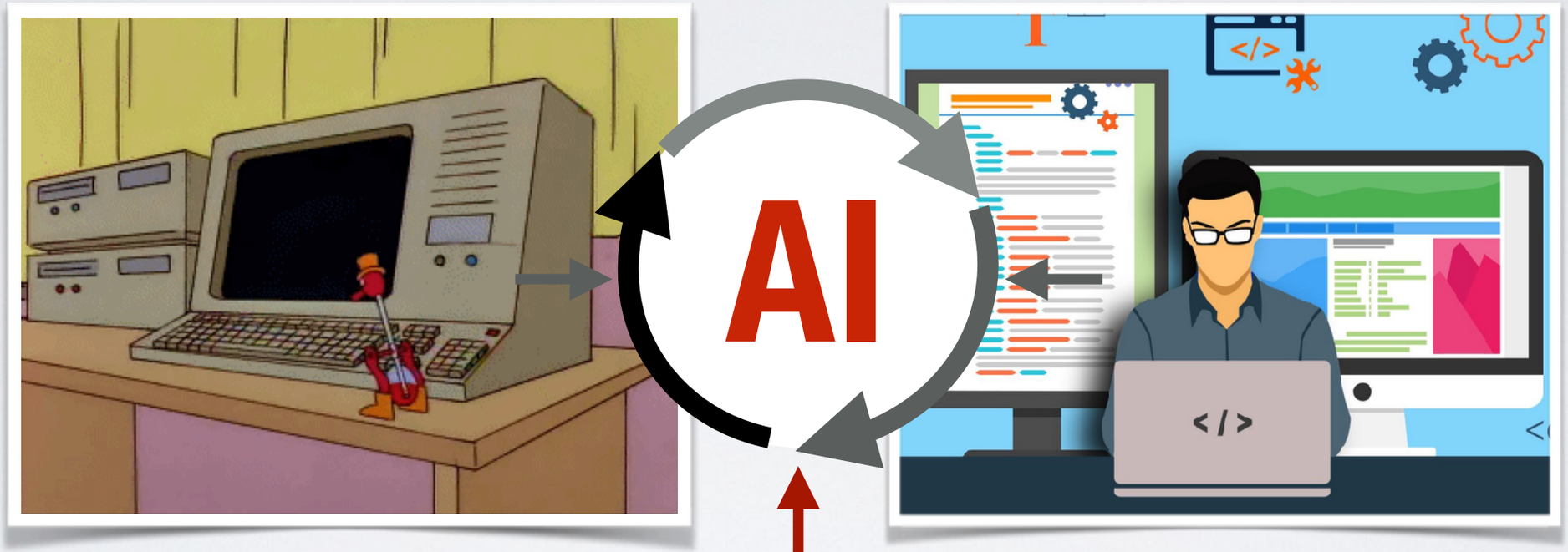
Search-Based Software Testing



Search-Based Software Testing



Automated Test Design and Execution



Random Fuzzer

Human Developers
(Are the tests good?)

Fitness
Function

The Five Types of Artificial Intelligence

TU Delft

[Domingos2015 “The Master Algorithm”]

Tribe	Origin	Master Algorithm
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Back-Propagation
Evolutionary	Evolutionary biology	Evolutionary Algorithms
Bayesian	Statistics	Probabilistic inference
Analogizers	Psychology	Kernel machines

Formal Methods

Neural Network

Regression

Support Vector

The Five Types of Artificial Intelligence

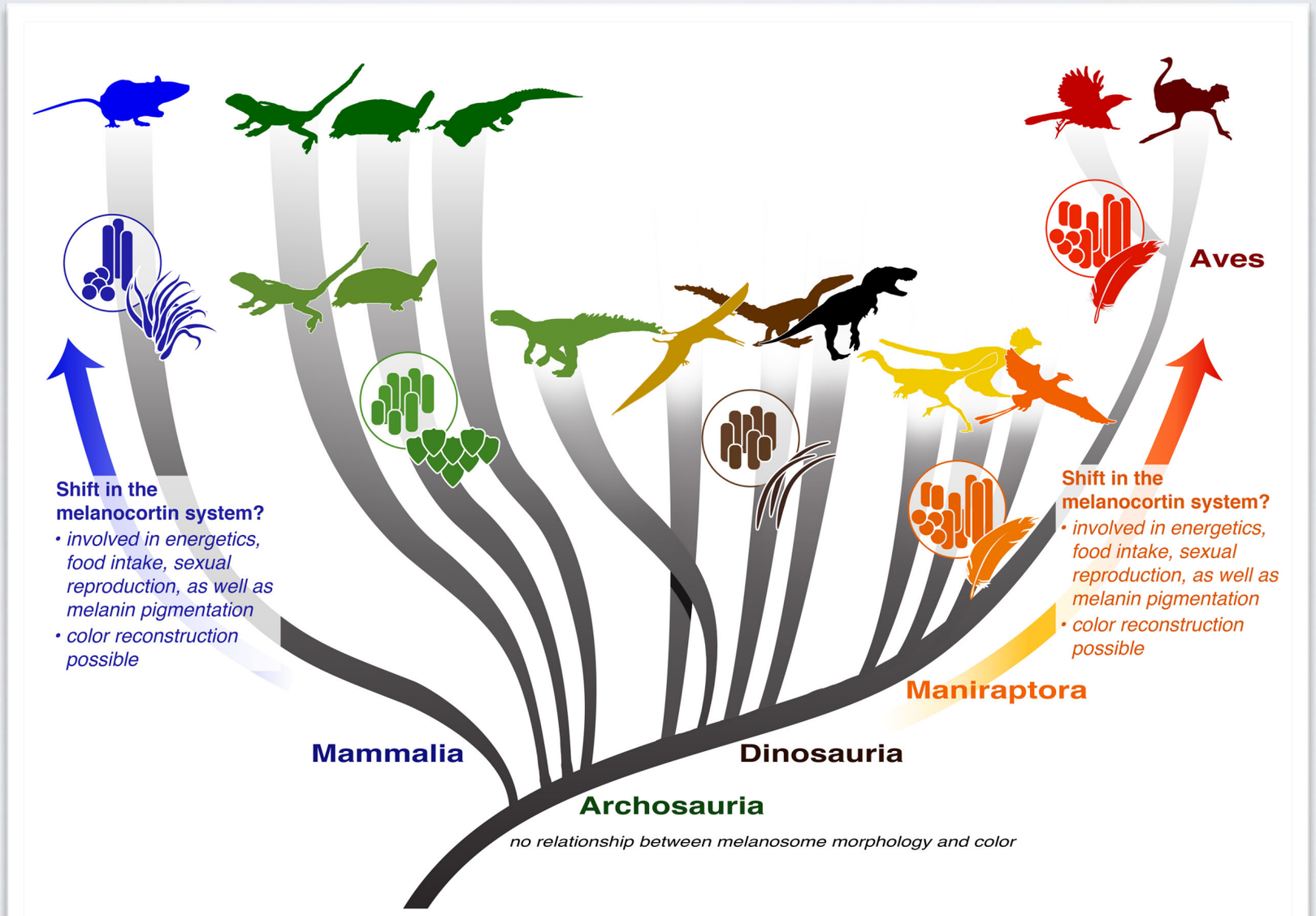
[Domingos2015 “The Master Algorithm”]

Tribe	Origin	Master Algorithm
Symbolists	Logic, philosophy	Inverse deduction
Connectionists	Neuroscience	Back-Propagation
Evolutionary	Evolutionary biology	Evolutionary Algorithms
Bayesian	Statistics	Probabilistic inference
Analogizers	Psychology	Kernel machines

Genetic Algorithms

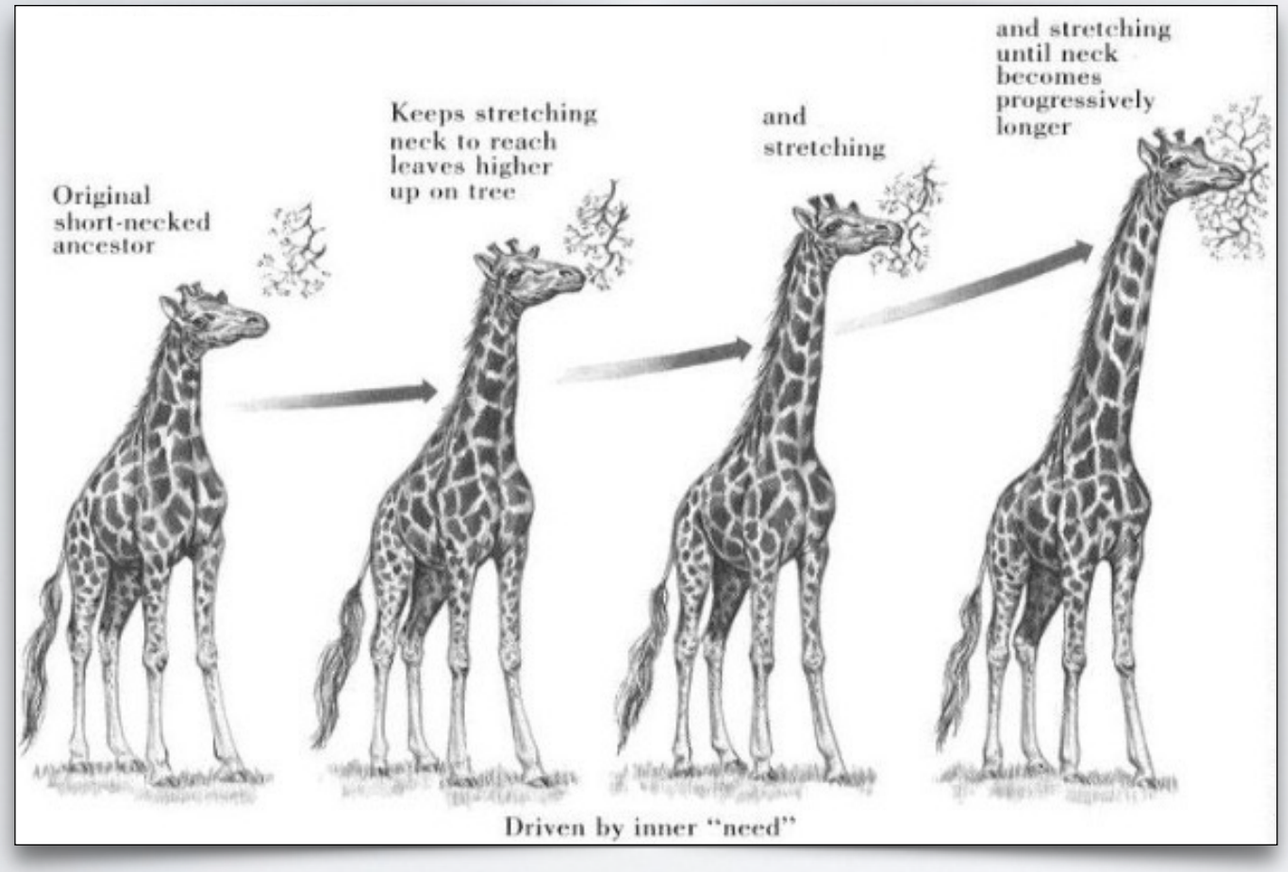
Genetic Algorithms

Genetic Algorithm: search algorithm inspired by evolution theory



Natural Evolution

Lamarck's Giraffe



Natural selection: Individuals that best fit the natural environment survive (worst die)

Reproduction: survived individuals generate offspring (next generation)

Mutation: offspring inherits properties of its parents (with some mutations)

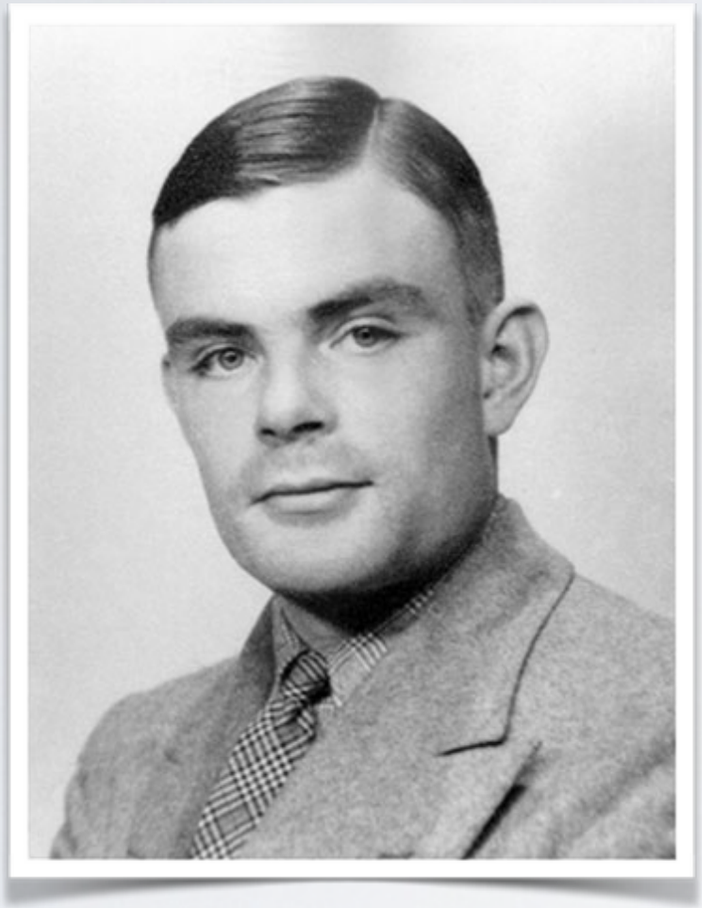
Iteration: generation by generation the new offspring fits better the environment than its ancestor

Genetic Algorithms

John Henry Holland

In 1975 he wrote the ground-breaking book on genetic algorithms, "Adaptation in Natural and Artificial Systems"

Origin of SBST



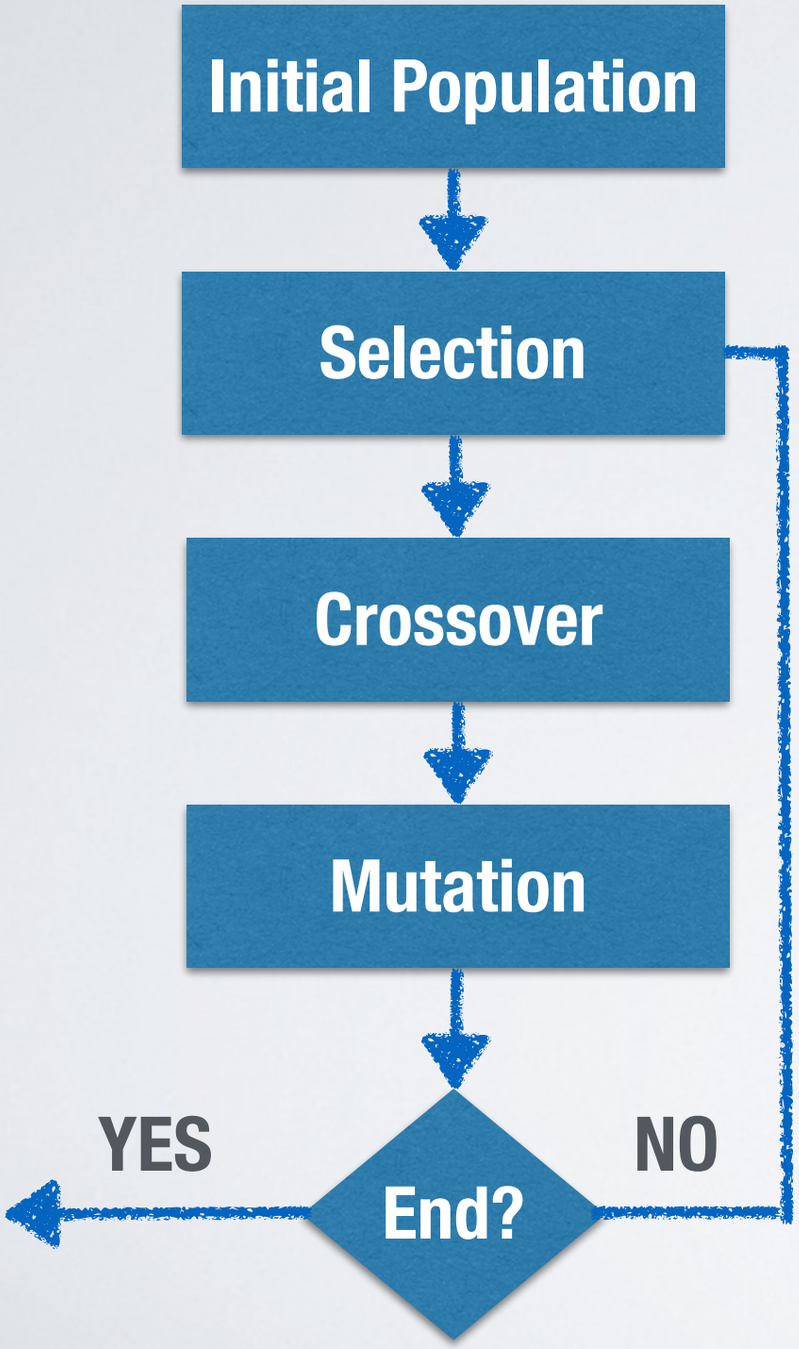
Alan Turing “Computing Machinery and Intelligence”, 1950

"Why not rather try to produce one [a program] which simulates the child’s [mind]?”

"We cannot expect to find a good child machine at the first attempt. […] There is an obvious connection between this process and evolution, by the identifications:

- Structure of the child machine = hereditary material
- Changes of the child machine = mutation,
- Natural selection = judgment of the experimenter

Genetic Algorithms



Natural selection: Individuals that best fit the natural environment survive (worst die)

Reproduction: survived individuals generate offspring (next generation)

Mutation: offspring inherits properties of the parents (with some mutations)

Iteration: generation by generation, the new offspring fits better the environment than their parents

Running Example 2

```
class Triangle {
   private double side1, side2, side3;
   private String type = “NOT_A_TRIANGLE”;

   public Triangle (double a, double b, double c){…}
   private void checkRightAngle() {…}
   public void computeTriangleType() {…}
   private boolean isTriangle() {…}
}
```

Running Example 2

```
class Triangle {
   private double side1, side2, side3;
   private String type = “NOT_A_TRIANGLE”;

   public Triangle (double a, double b, double c){
      side1 = a;
      side2 = b;
      side3 = c;
   }

   private void checkRightAngle() {…}
   public void computeTriangleType() {…}
   private boolean isTriangle() {…}
}
```

Running Example 2

```
class Triangle {
   private double side1, side2, side3;
   private String type = “NOT_A_TRIANGLE”;

   public Triangle (double a, double b, double c){…}

   private void checkRightAngle() {
     if (side1*side1 + side2*side2 == side3*side3)
        type = “RIGHT_ANGLE”;
     else if (side1*side1 + side3*side3 == side2*side2)
        type = “RIGHT_ANGLE”;
     else if (side3*side3 + side2*side2 == side1*side1)
        type = “RIGHT_ANGLE”;
     else
        type = “SCALENE”;
   }

   public void computeTriangleType() {…}
   private boolean isTriangle() {…}
}
```

Running Example 2

```
class Triangle {
   private double side1, side2, side3;
   private String type = “NOT_A_TRIANGLE”;
   public Triangle (double a, double b, double c){…}
   private void checkRightAngle() {…}

   public void computeTriangleType(){…}

   private boolean isTriangle() {
     if (side1<=0)
        return false;
     if (side2<=0)
        return false;
     if (side3<=0)
        return false;

     return true;
   }
}
```

Running Example 2

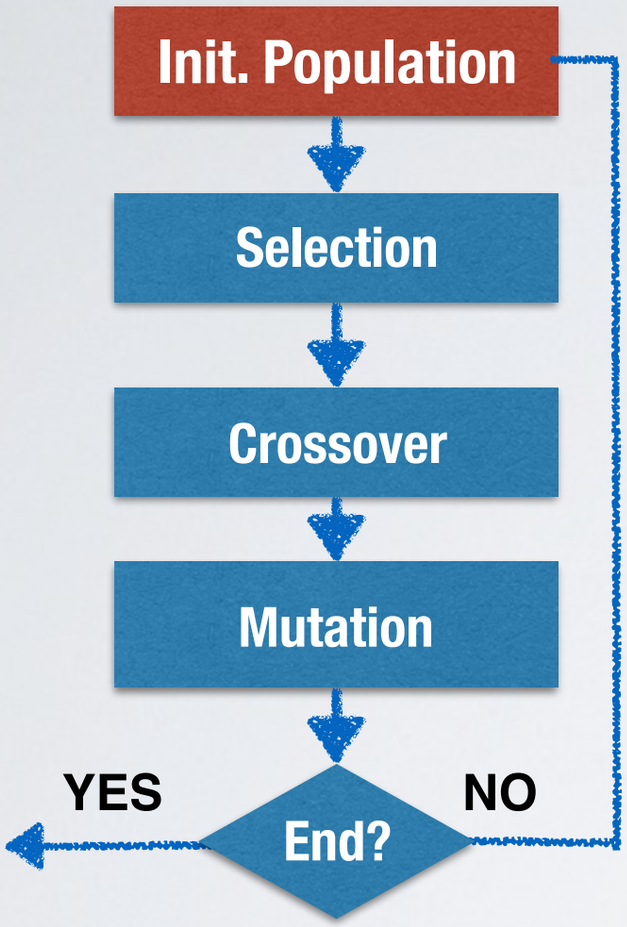
```
class Triangle {
    private double side1, side2, side3;
    private String type = “NOT_A_TRIANGLE”;
    public Triangle (double a, double b, double c){…}
    private void checkRightAngle() {…}

    public void computeTriangleType(){
      if (isTriangle()){
           if (side1 == side2) {
               if (side2 == side3)
                    type = "EQUILATERAL";
               else
                    type = "ISOSCELES";
           } else {
               if (side1 == side3) {
                    type = "ISOSCELES";
               } else {
                    if (side2 == side3)
                         type = “ISOSCELES”;
                    else
                         checkRightAngle();
               }
           }
      }// if isTriangle()
    }
    private boolean isTriangle() {…}
}
```

Running Example 2

```
class Triangle {
   private double side1, side2, side3;
   private String type = “NOT_A_TRIANGLE”;

   public Triangle (double a, double b, double c){…}
   private void checkRightAngle() {…}
   public void computeTriangleType() {…}
   private boolean isTriangle() {…}
}
```

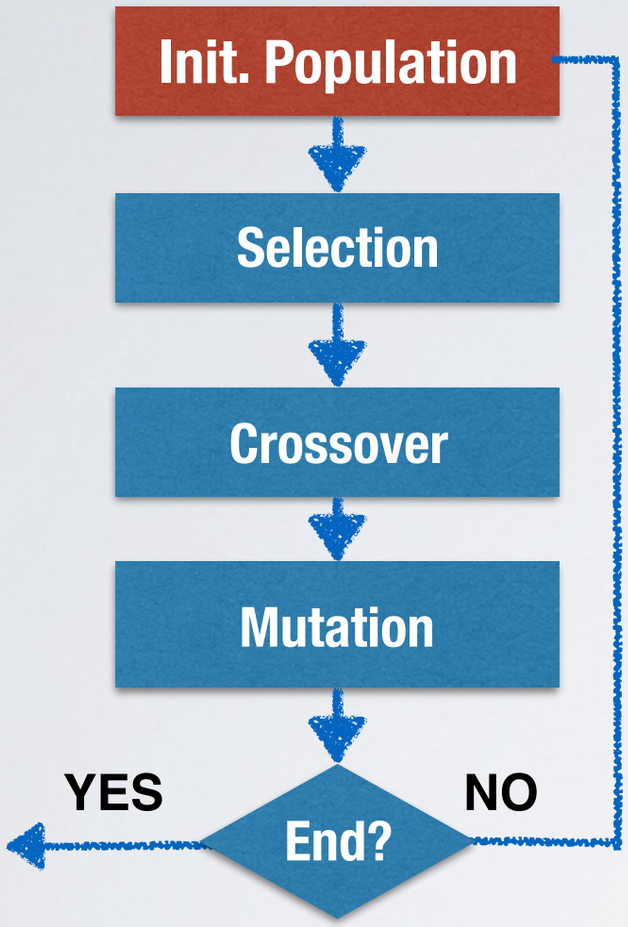
GOAL: Automatic generation of test cases using GAs in order to achieve the maximum statement coverage

Starting GAs



The initial population is a set of randomly generated test cases. We can apply the same procedure used in Random Testing for the initial population.

Starting GAs



Example of randomly generated initial population:

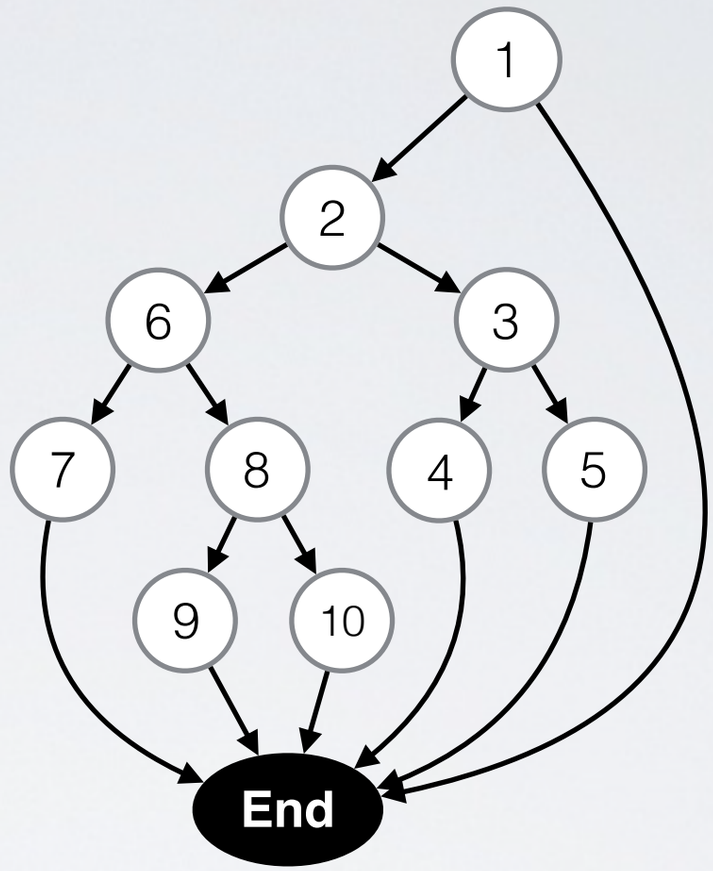
Pop8 = { x1 = (2,2,3),
x2 = (2,3,5),
x3 = (-2,3,6),
x4 = (2,3,7),
x5 = (2,2,3),
x6 = (3,4,5),
x7 = (3,5,7),
x8 = (6,8,4) }

N.B.: In our running example we have only one method with input parameter (the constructor). In the following, we will use only the input vector to denote the test case

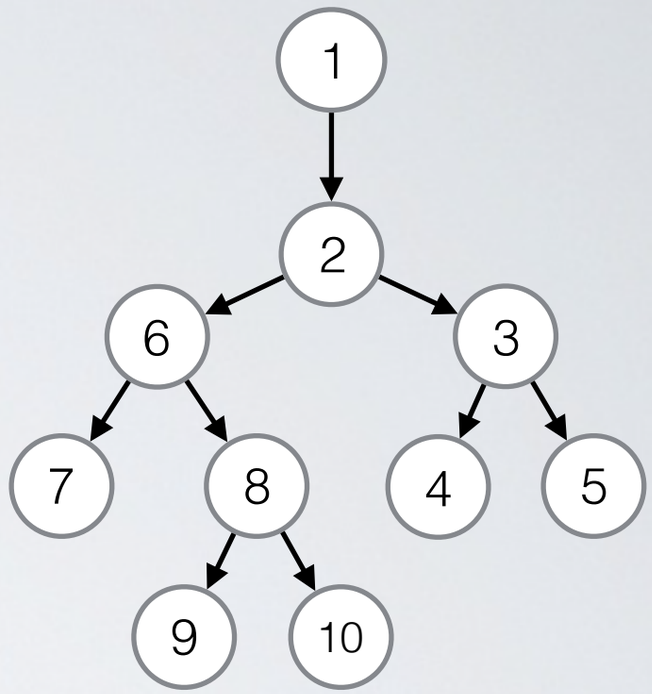
How Fit are Our Random Tests?

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```



Control flow graph

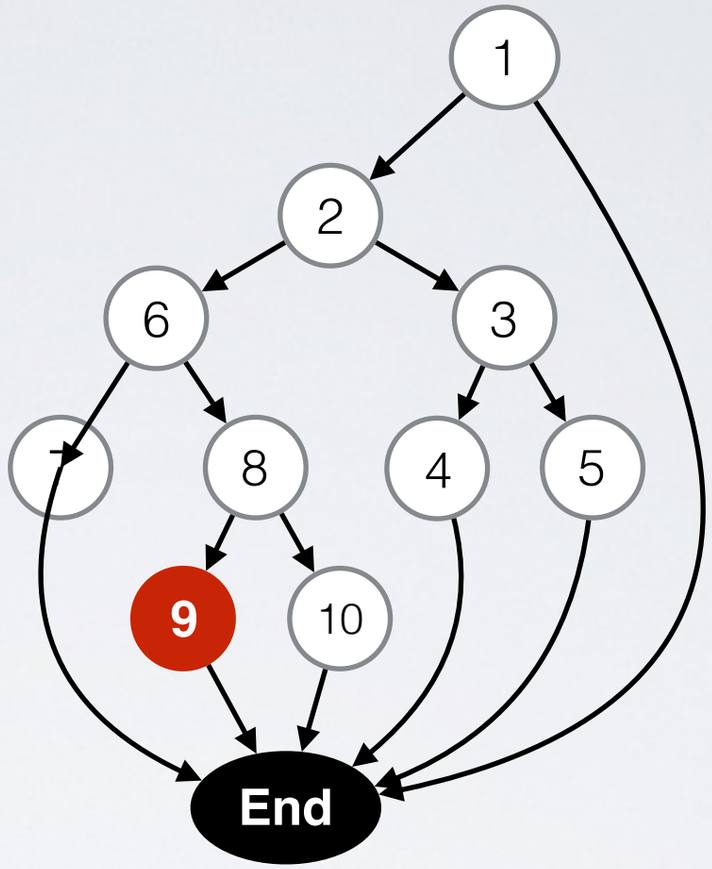


Dependency graph

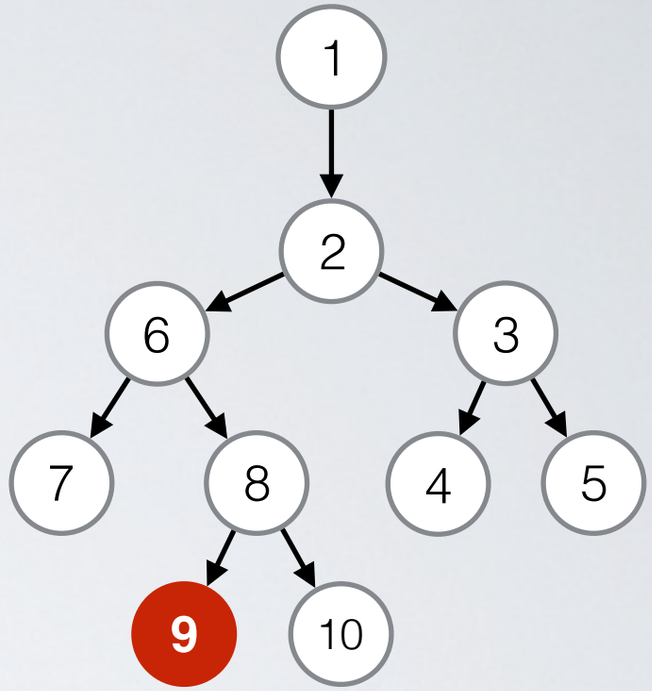
How Fit are Our Random Tests?

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.         if (side2 == side3)
4.             type = "EQUILATERAL";
           else
5.             type = "ISOSCELES";
      } else {
6.         if (side1 == side3) {
7.             type = "ISOSCELES";
           } else {
8.             if (side2 == side3)
9.  target ⟶   type = “ISOSCELES”;
               else
10.               checkRightAngle();
               }
           }
      }// if isTriangle()
  }}
```



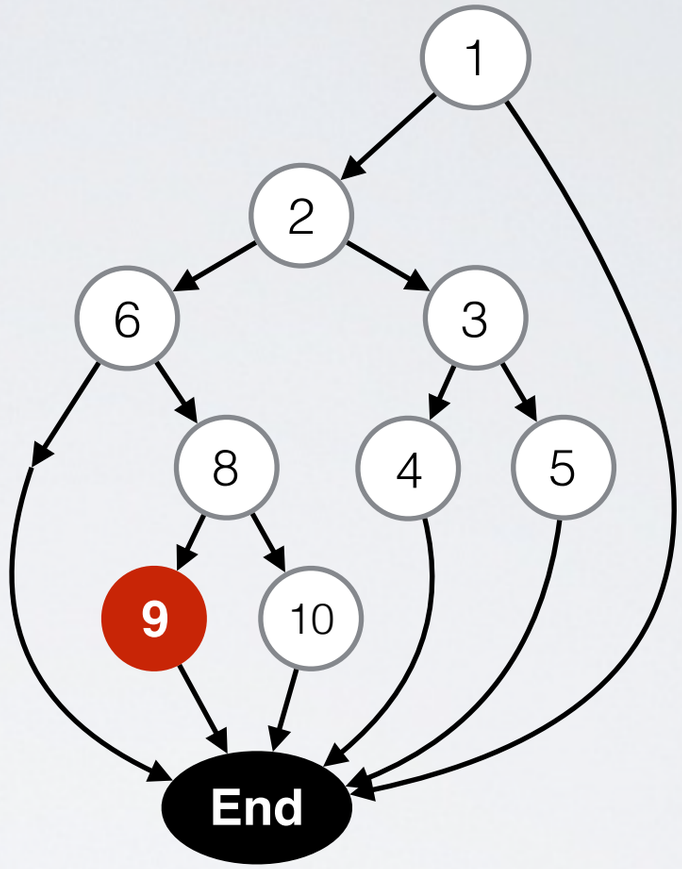
Control flow graph



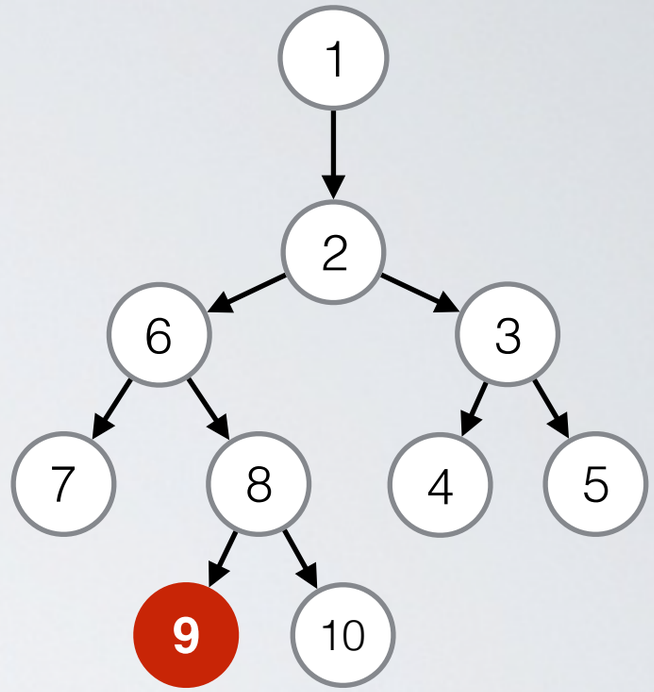
Dependency graph

How Fit are Our Random Tests?

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.         if (side2 == side3)
4.             type = "EQUILATERAL";
           else
5.             type = "ISOSCELES";
      } else {
6.         if (side1 == side3) {
7.             type = "ISOSCELES";
           } else {
8.             if (side2 == side3)
9.  Target→     type = “ISOSCELES”;
               else
10.               checkRightAngle();
               }
           }
      }// if isTriangle()
  }}
```



Control flow graph



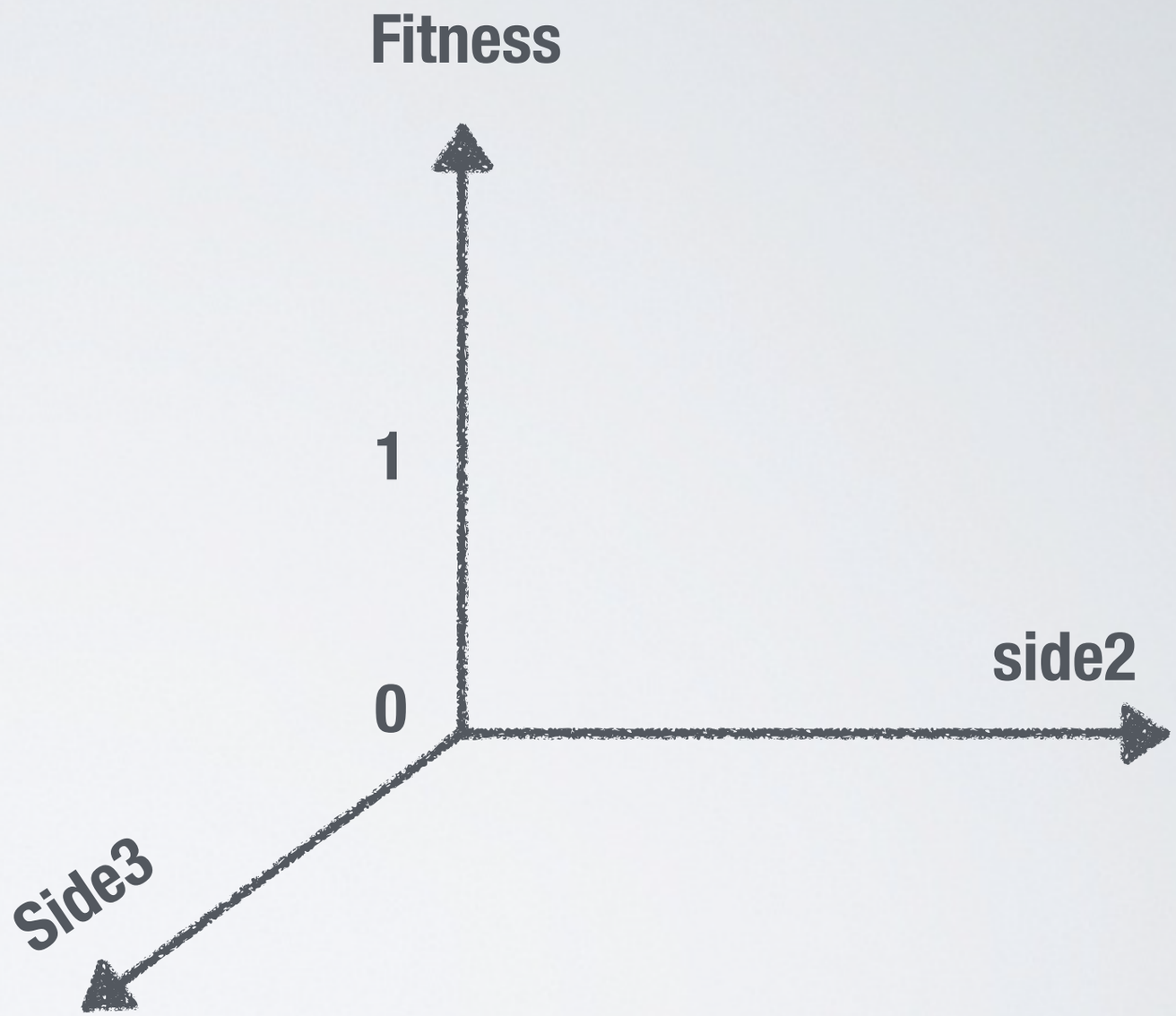
Dependency graph

A simple heuristic (or **fitness function**) could be:

$$f(side2; side3) = \begin{cases} 0 \text{ if side2 == side3)} \\ 1 \text{ if side2 != side3)} \end{cases}$$

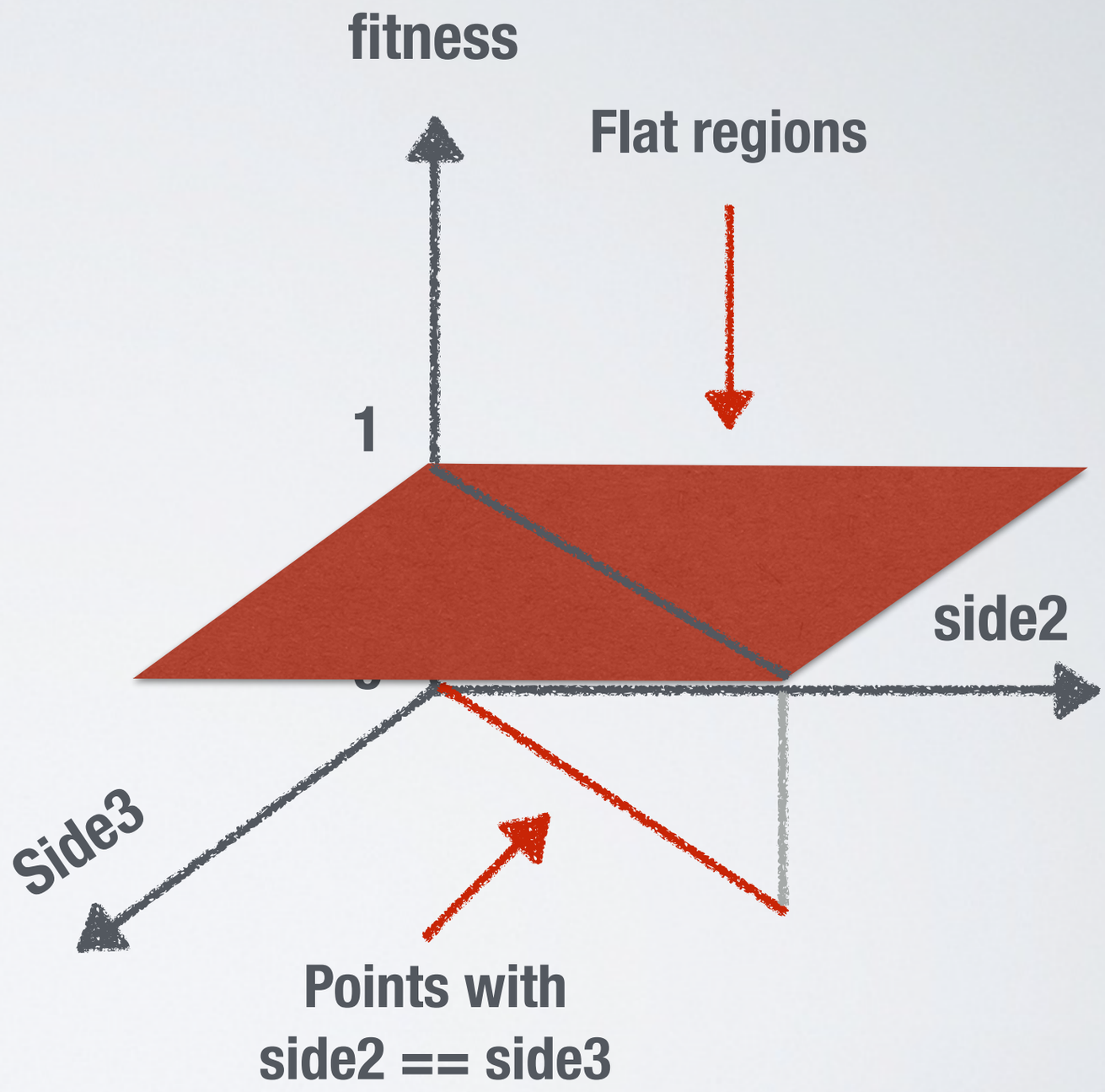
Fitness Function: Attempt 1

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.         if (side2 == side3)
4.             type = "EQUILATERAL";
           else
5.             type = "ISOSCELES";
      } else {
6.         if (side1 == side3) {
7.             type = "ISOSCELES";
           } else {
8.             if (side2 == side3)
9.  Target→     type = “ISOSCELES”;
               else
10.                checkRightAngle();
               }
           }
      }// if isTriangle()
  }}
```



Fitness Function: Attempt 1

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.         if (side2 == side3)
4.             type = "EQUILATERAL";
           else
5.             type = "ISOSCELES";
      } else {
6.         if (side1 == side3) {
7.             type = "ISOSCELES";
           } else {
8.             if (side2 == side3)
9.  Target→        type = “ISOSCELES”;
               else
10.               checkRightAngle();
               }
           }
      }// if isTriangle()
  }}
```



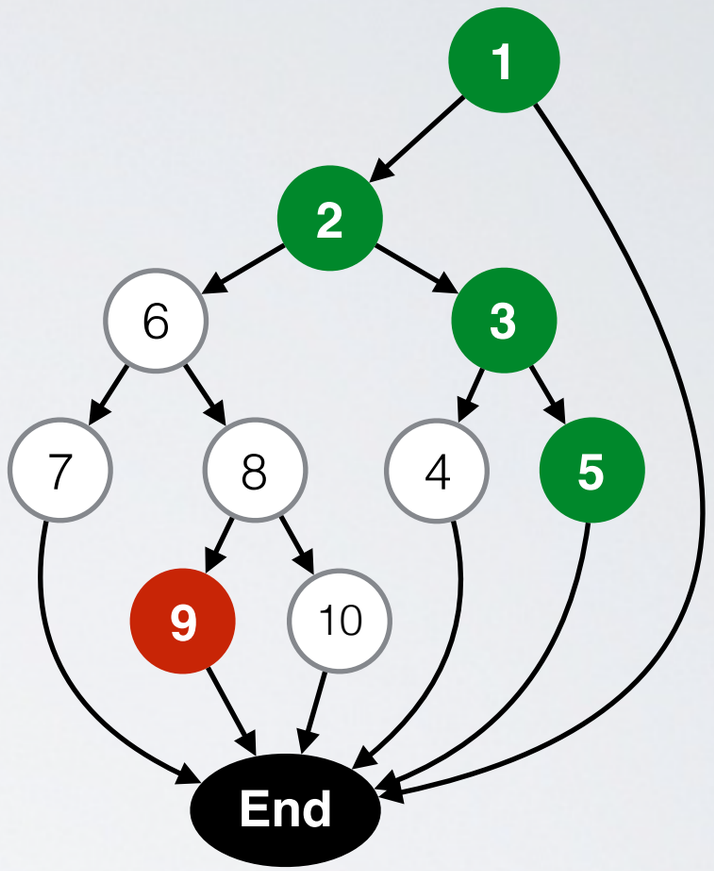
Flag problem: flat fitness function that does not provide any guidance

Fitness Function: Attempt 2

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

Target

Control flow graph



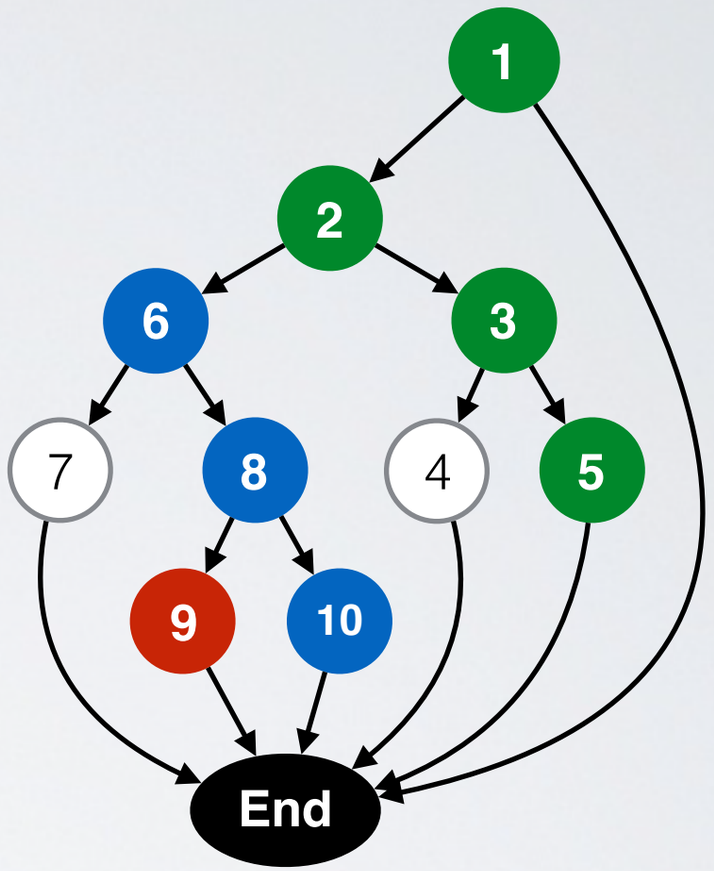
x1 = (2, 2, 3) **Path(x1) = <1, 2, 3, 5>**

Fitness Function: Attempt 2

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
     }// if isTriangle()
  }}
```

Target

Control flow graph



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5>

x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10>

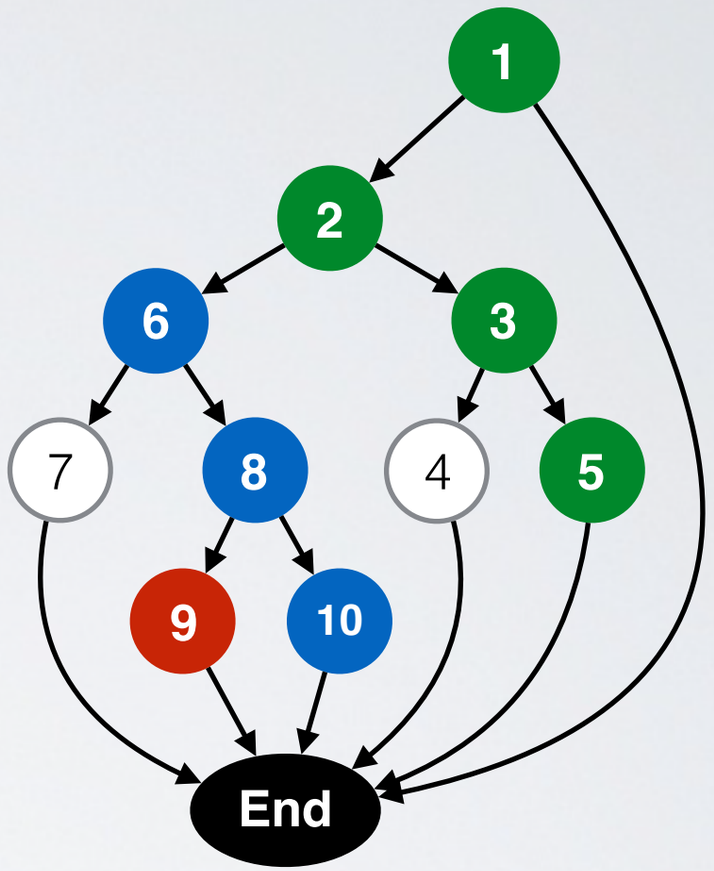
Fitness Function: Attempt 2

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
      }// if isTriangle()
  }}
```

Target

Control flow graph



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5>

x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10>

x3 = (-2,10,8) Path(x3) = <1>

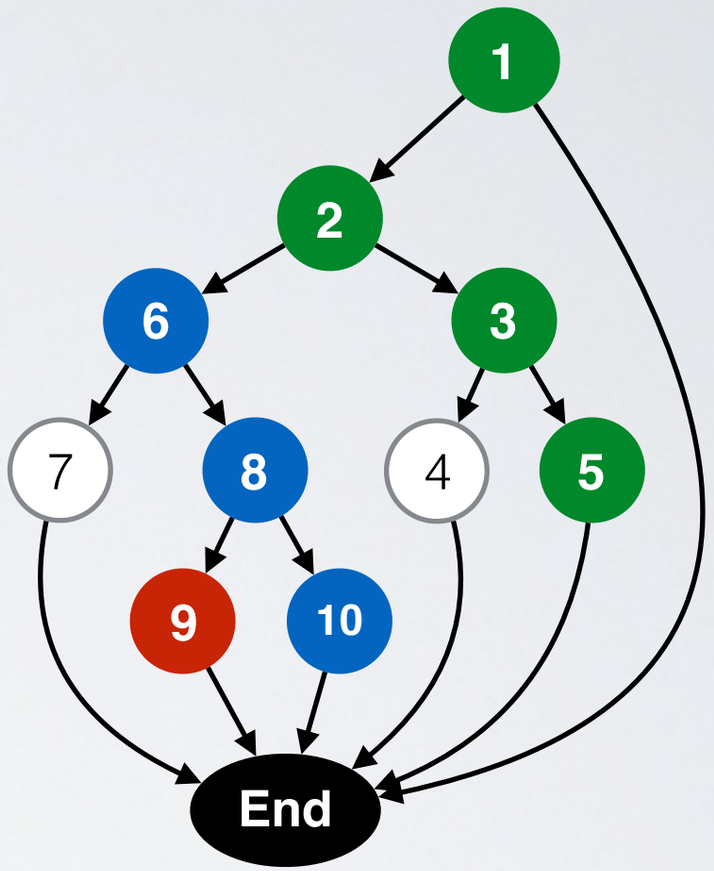
Fitness Function: Attempt 2

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
      }// if isTriangle()
  }}
```

Target

Control flow graph



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5>

x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10>

x3 = (-2,10,8) Path(x3) = <1>

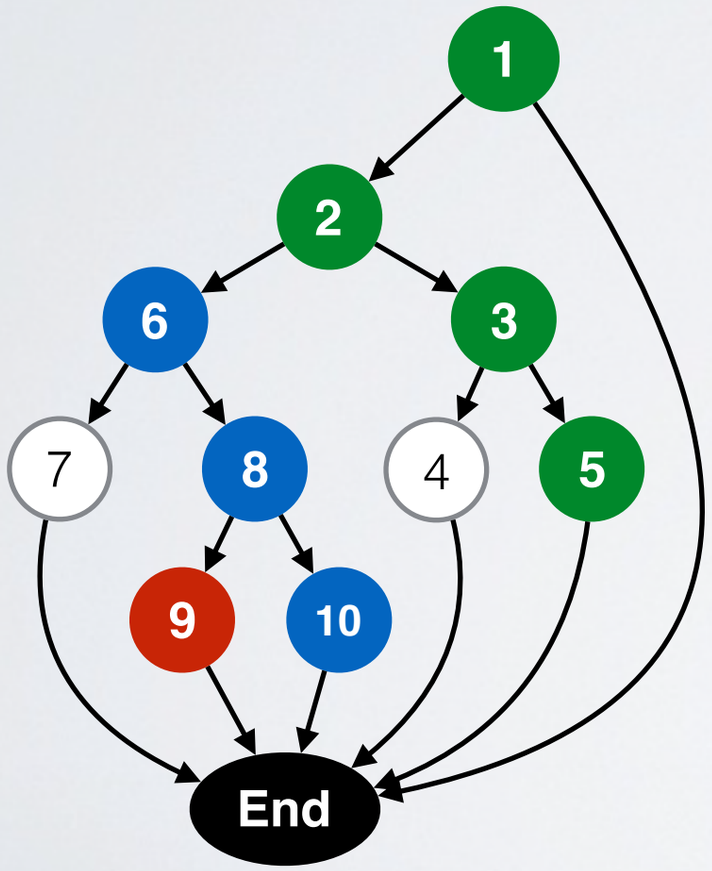
What is the closest TC to cover the statement 9?

Approach Level

TU Delft

Approach_level(P(x), t)

Given the execution trace obtained by running program P with test case x, the approach level is the minimum number of control nodes between an executed statement and the coverage target t.



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5>

x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10>

Approach_Level(x1) = ?

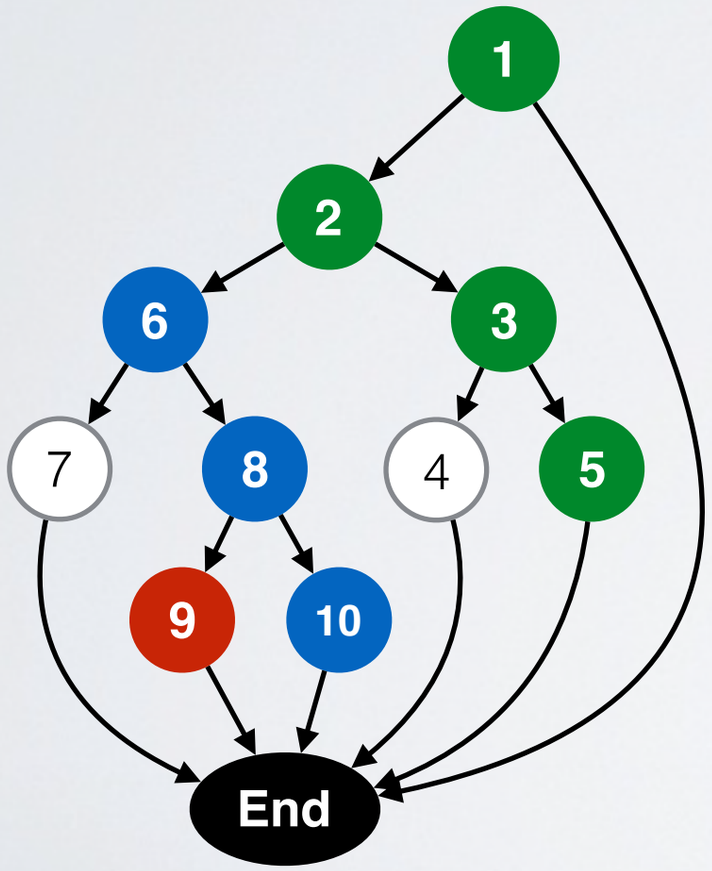
Approach_Level(x2) = ?

Approach Level

TU Delft

Approach_level(P(x), t)

Given the execution trace obtained by running program P with test case x, the approach level is the minimum number of control nodes between an executed statement and the coverage target t.



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5>

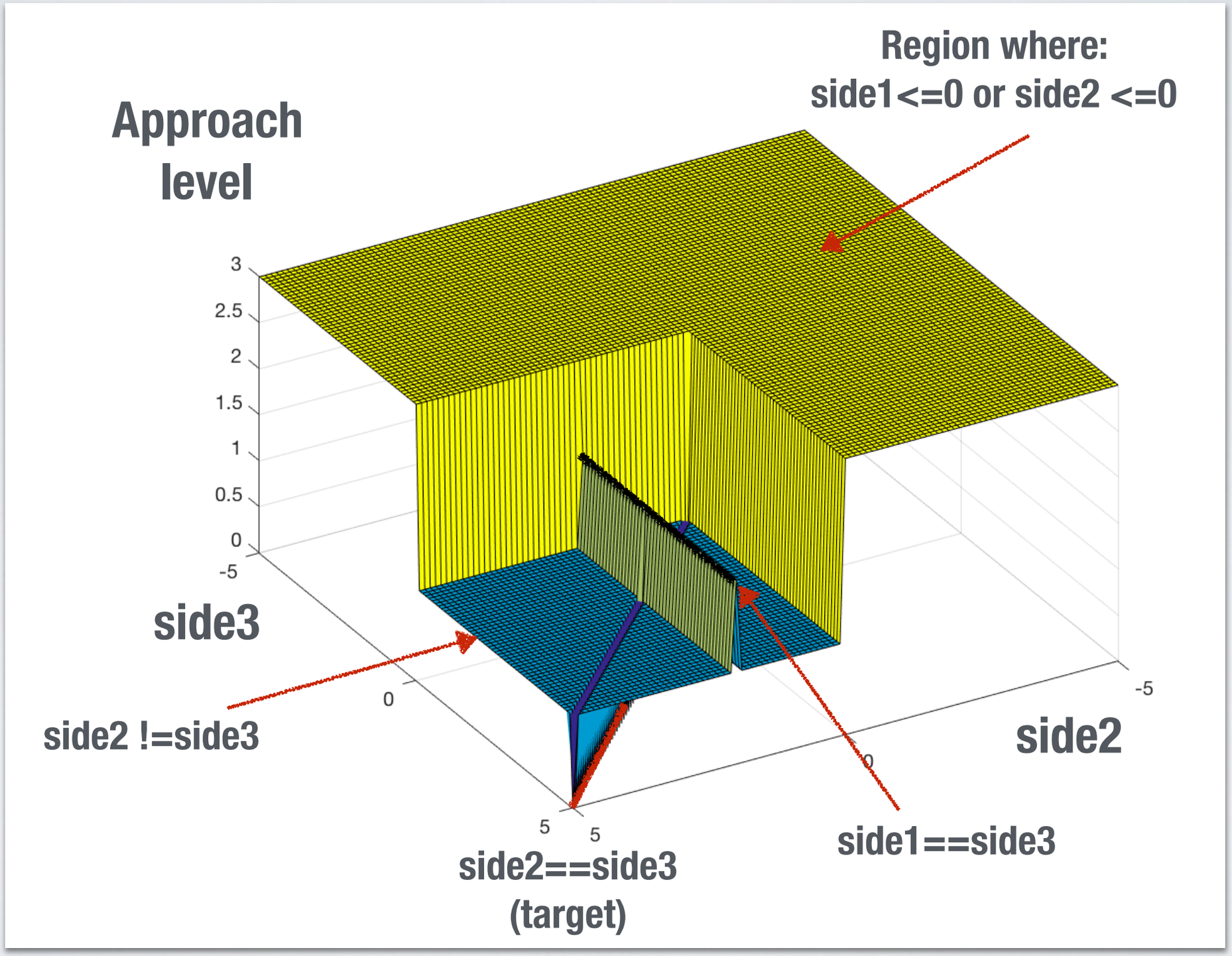
x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10>

Approach_Level(x1) = 2

Approach_Level(x2) = 0

Dijkstra's Algorithm for Shortest Path

Approach Level



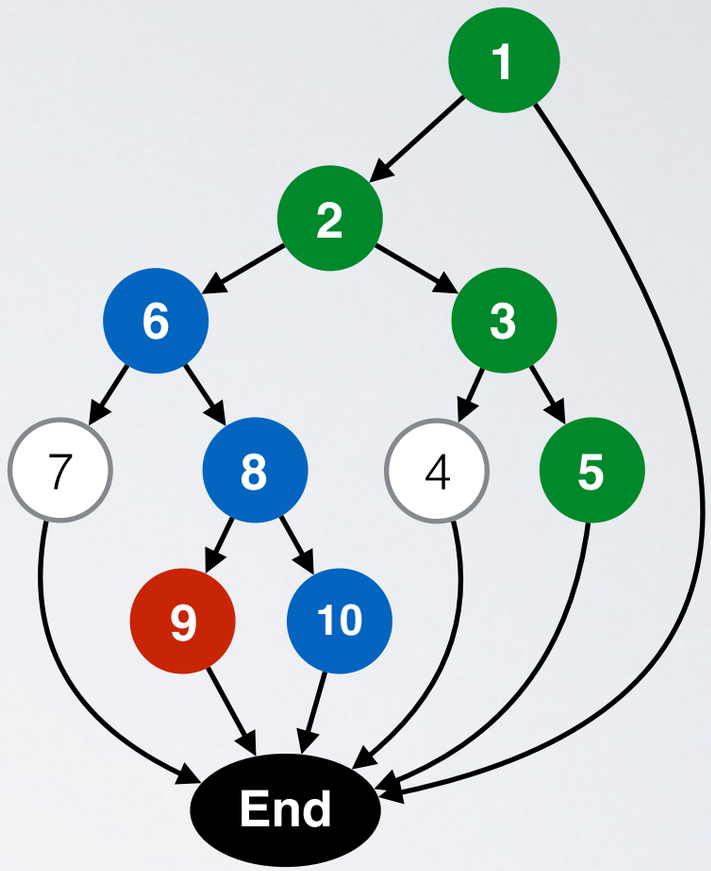
Fitness Function: Attempt 3

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

Target

Control flow graph



x1 = (2, 2, 3) Path(x1) = <1, 2, 3, 5> AL=2

x2 = (2, 3, 5) Path(x2) = <1, 2, 6, 8, 10> AL=0

x3 = (2, 3, 10) Path(x3) = <1, 2, 6, 8, 10> AL=0

What is the closest TC to cover the statement 9?

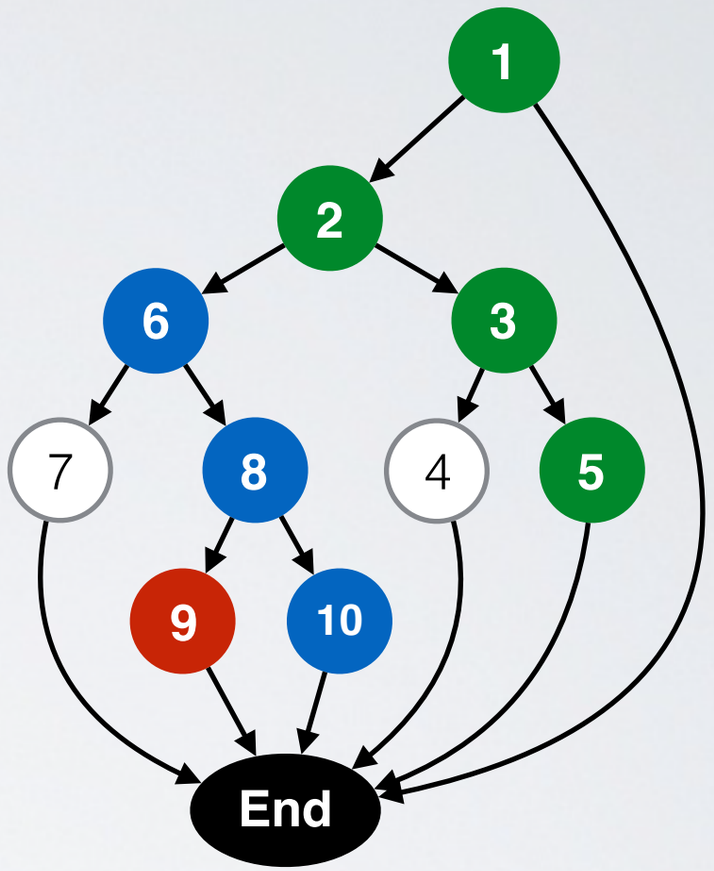
Fitness Function: Attempt 3

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
      }// if isTriangle()
  }}
```

Target

Control flow graph



x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	-
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	if (3==5)
x3 = (2, 3, 10)	Path(x3) = <1, 2, 6, 8, 10>	if (2==10)

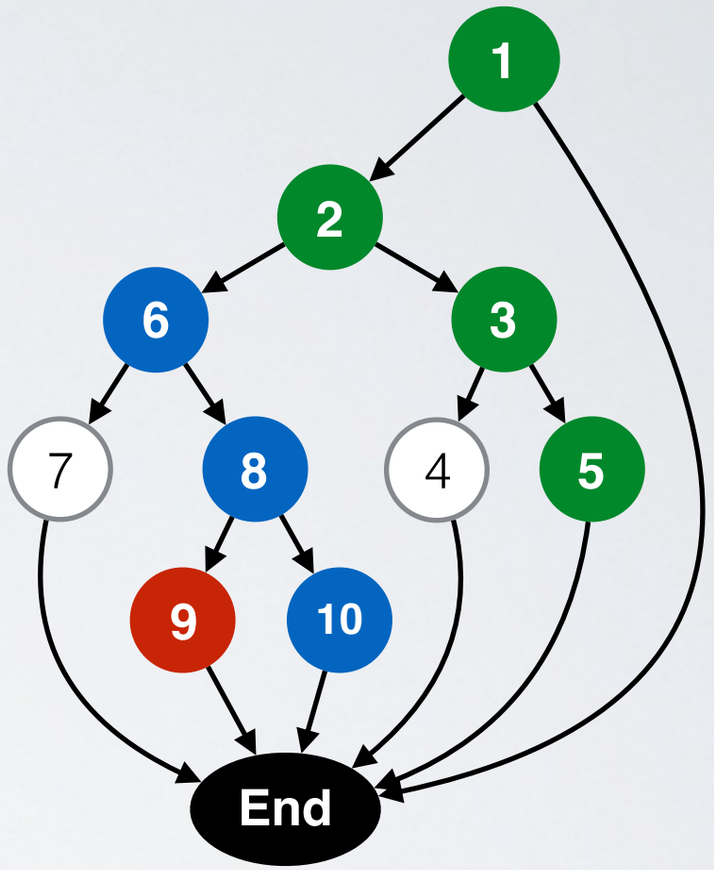
Fitness Function: Attempt 3

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
      }// if isTriangle()
  }}
```

Target

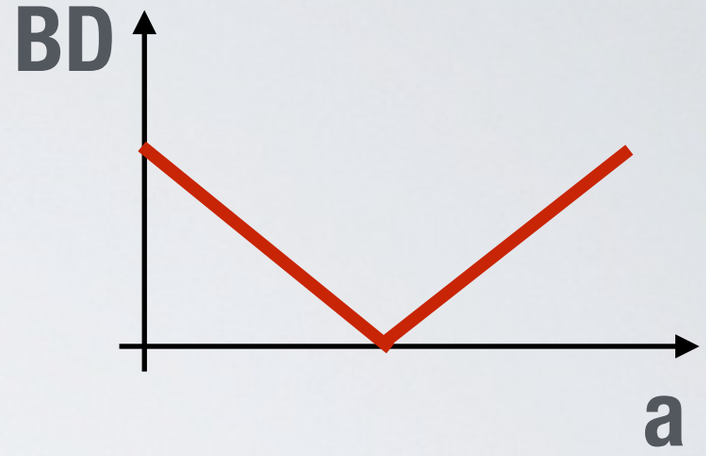
Control flow graph



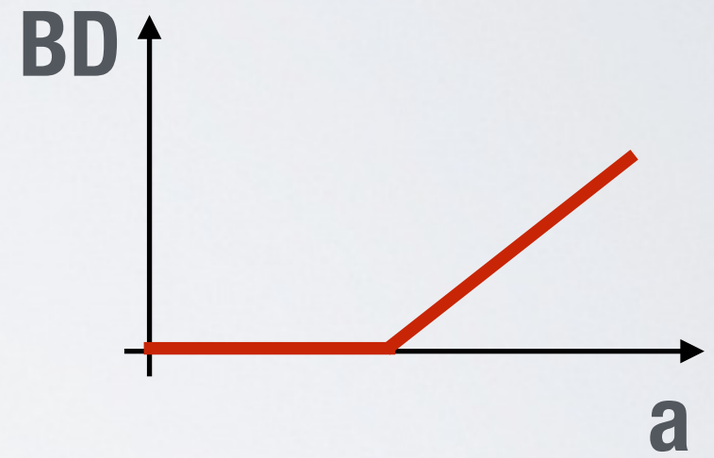
x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	-
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	abs(3-5) =2
x3 = (2, 3, 10)	Path(x3) = <1, 2, 6, 8, 10>	abs(2-10)=8

Branch Distance

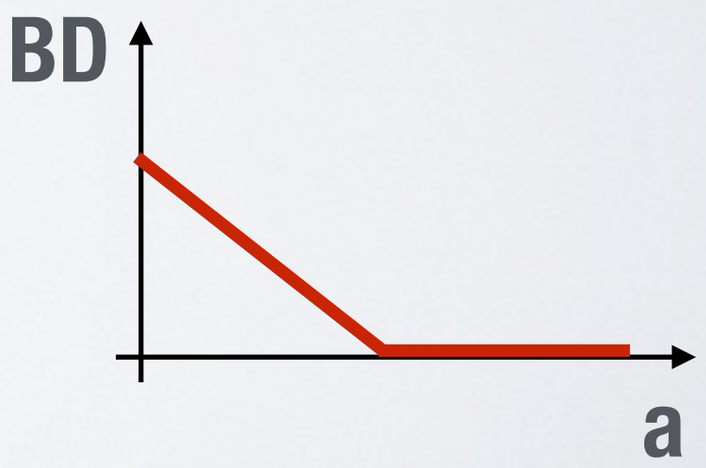
If (a == b) ⟶ abs(a-b)



If (a <= b) ⟶ 0 if a<=b
abs(a-b) otherwise



If (a => b) ⟶ 0 if >=b
abs(a-b) otherwise



Branch Distance

TU Delft

Branch_distance(P(x), t)

Given the first control node where the execution diverges from the target t, the predicate at such node is converted to a distance (from taking the desired branch), normalised between 0 and 1.

Such a distance measures how fare the test case is from taking the desired branch. For boolean and numerical variables a, b:

Condition c = atomic predicate	Distance BD(c) = d / (d + 1)
a	d = {0 if a == true; K otherwise}
!a	d = {K if a == true; 0 otherwise}
a == b	d = {0 if a == b; abs(a - b) + K otherwise}
a != b	d = {0 if a != b; K otherwise}
a < b	d = {0 if a < b; a - b + K otherwise}
a <= b	d = {0 if a <= b; a - b + K otherwise}
a > b	d = {0 if a > b; b - a + K otherwise}
a >= b	d = {0 if a >= b; b - a + K otherwise}

Branch Distance for Strings

TU Delft

Branch_distance(P(x), t)

For string variables a and b, the branch distance is computed using the following rules:

Condition c = atomic predicate	Distance BD(c) = d / (d + 1)
a == b	d = {0 if a == b; edit_dist(a, b) + K otherwise}
a != b	d = {0 if a != b; K otherwise}
a < b	d = {0 if a < b; a[j] - b[j] + K otherwise}
a <= b	d = {0 if a <= b; a[j] - b[j] + K otherwise}
a > b	d = {0 if a > b; b[j] - a[j] + K otherwise}
a >= b	d = {0 if a >= b; b[j] - a[j] + K otherwise}

Example of edit distance: edit_dist(“abcd”, “abbb”)=2

Branch Distance

Branch distance rules for composite predicate

Condition c = composite predicate	Distance BD(c) = d / (d + 1)
!p	Negation is propagated inside p
p & q	d = d(p) + d(q)
p | q	d = min(d(p), d(q))
p XOR q = p & !q | !p & q	d = min(d(p)+d(!q), d(!p)+d(q))

How to normalise the branch distance?

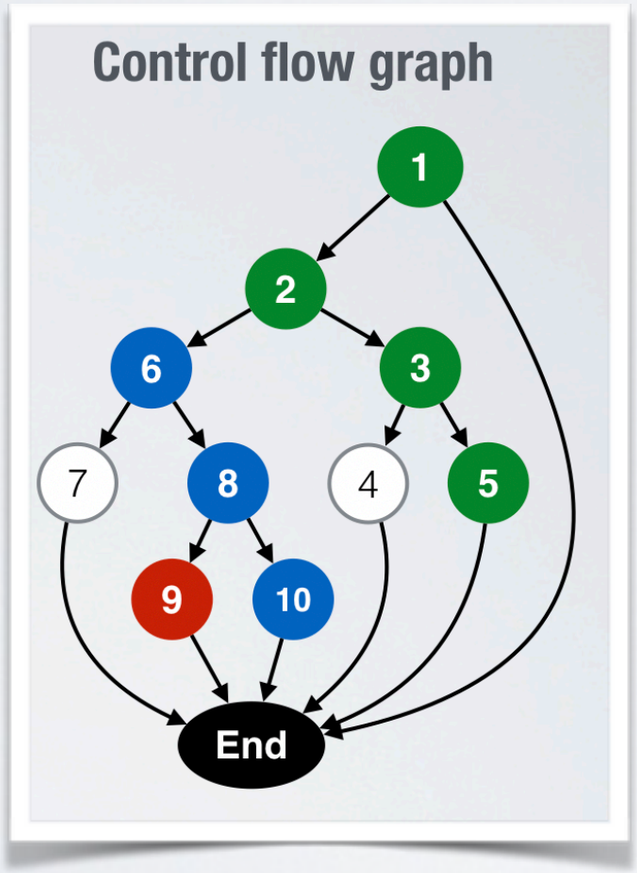
branch_distance(t) = d/(d+1)

where d is computed according to the distance rules reported in the previous tables

Fitness Function

For statement (or branch) coverage, given a specific coverage target t, a widely used fitness function (to be minimised) is:

$$f(x) = approach\_level(P(x), t) + branch\_distance(P(x),t)$$

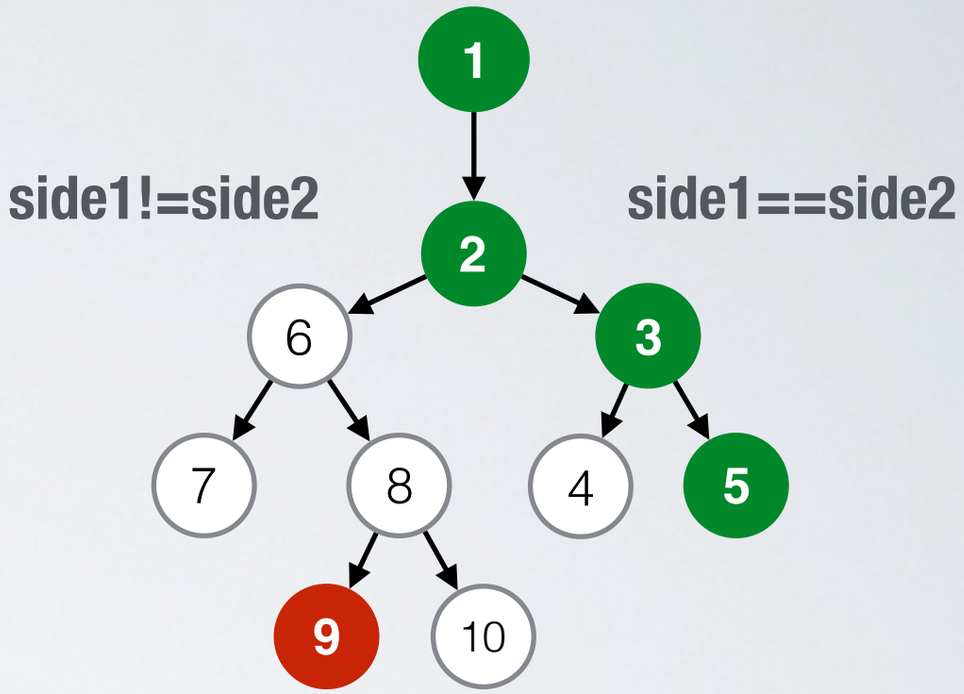


+

Fitness Function

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

Target



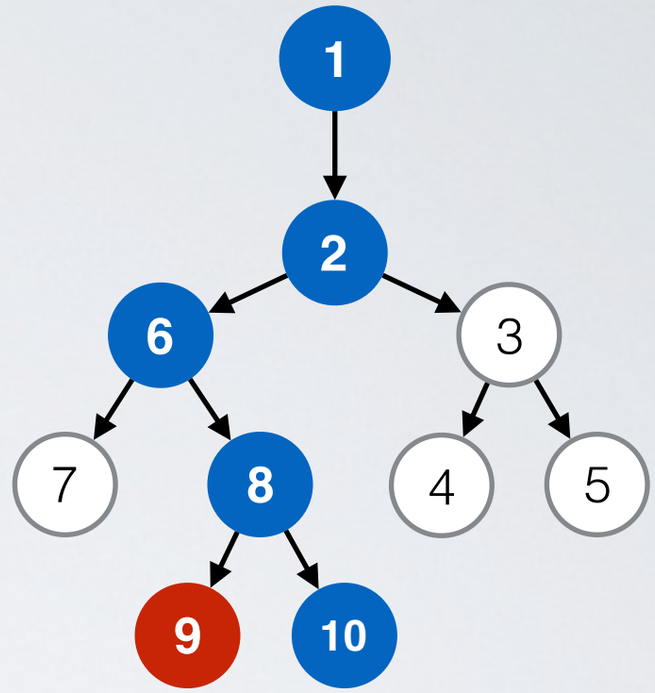
d(2 != 2) = 1
BD (2 != 2) = 1/ (1+1) = 0.5
f(x1) = 2 + 0.5 = 2.5

x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	AL=2	f = 2.5
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	AL=0	
x3 = (1, 2, 10)	Path(x3) = <1, 2, 6, 8, 10>	AL=0	

Fitness Function

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

Target



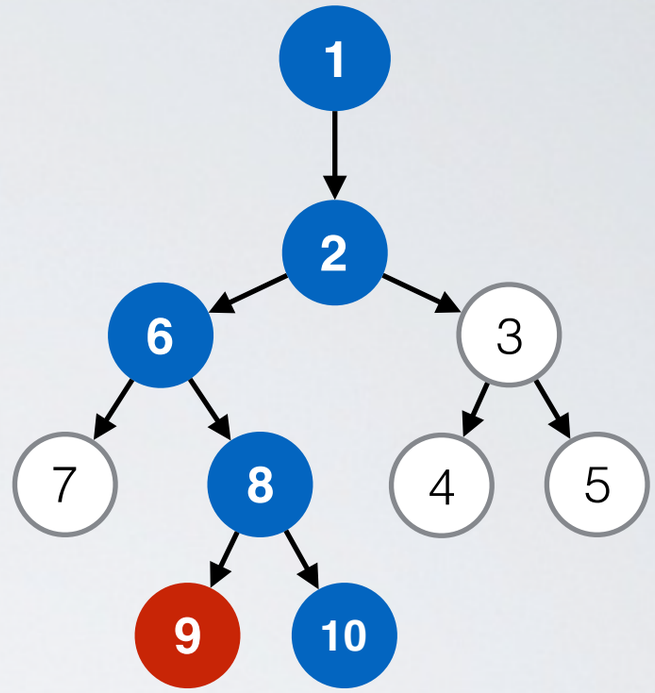
d(3 == 5) = abs(3-5) = 2
BD (3 == 5) = 2 / (2+1) = 0.66
f(Ch1) = 0 + 0.66 = 0.66

x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	AL=2	f = 2.5
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	AL=0	f = 0.66
x3 = (1, 2, 10)	Path(x3) = <1, 2, 6, 8, 10>	AL=0	

Fitness Function

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

Target



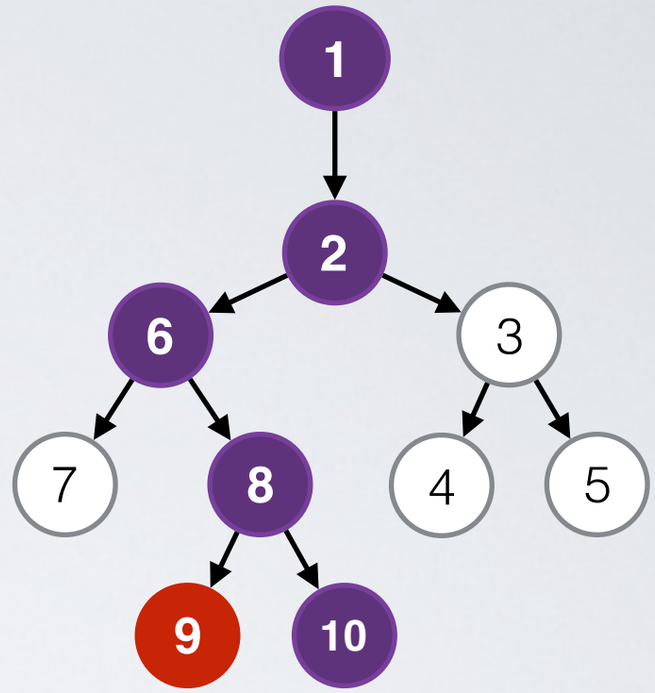
d(3 == 5) = abs(3-5) = 2
BD (3 == 5) = 2 / (2+1) = 0.66
f(Ch1) = 0 + 0.66 = 0.66

x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	AL=2	f = 2.5
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	AL=0	f = 0.66
x3 = (1, 2, 10)	Path(x3) = <1, 2, 6, 8, 10>	AL=0	f = ?

Fitness Function

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.        if (side2 == side3)
4.            type = "EQUILATERAL";
          else
5.            type = "ISOSCELES";
      } else {
6.        if (side1 == side3) {
7.            type = "ISOSCELES";
          } else {
8.            if (side2 == side3)
9.                type = “ISOSCELES”;
              else
10.               checkRightAngle();
              }
          }
      }// if isTriangle()
  }}
```

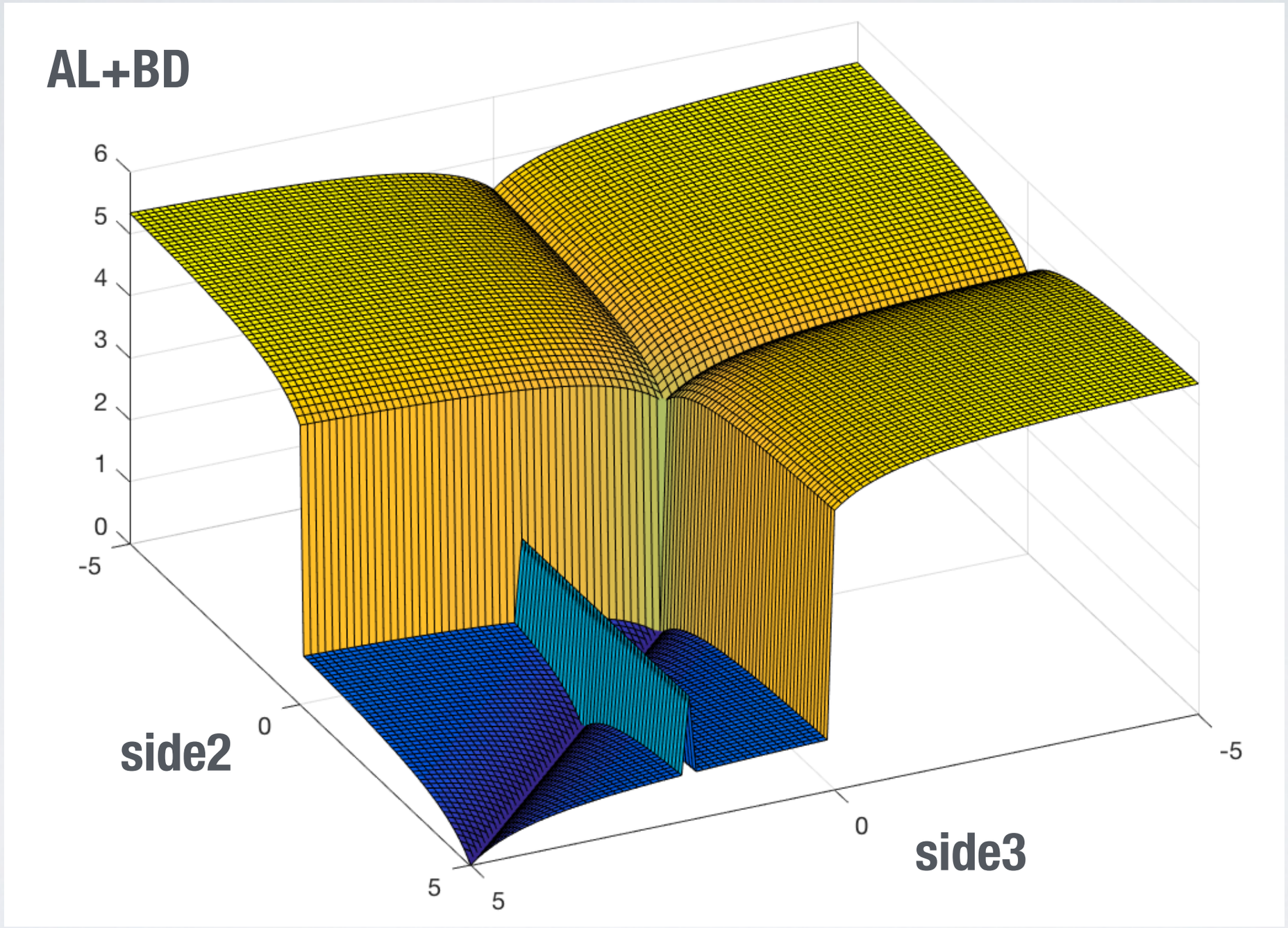
Target



d(2 == 10) = abs(2-10) = 8
BD (2 == 10) = 8 / (8+1) = 0.89
f(x3) = 0 + 0.89 = 0.89

x1 = (2, 2, 3)	Path(x1) = <1, 2, 3, 5>	AL=2	f = 2.5
x2 = (2, 3, 5)	Path(x2) = <1, 2, 6, 8, 10>	AL=0	f = 0.66
x3 = (1, 2, 10)	Path(x3) = <1, 2, 6, 8, 10>	AL=0	f = 0.89

Fitness Function

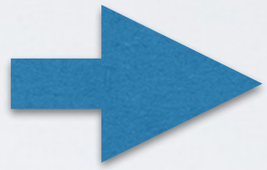


Tournament Selection

TU Delft

Tournament Selection

		Tournaments	Winners
x1 = (2,2,3)	f = 2.50	x2, x7	
x2 = (2,3,5)	f = 0.66	x1, x5	
x3 = (-2,3,6)	f = 3.66	x3, x8	
x4 = (2,3,7)	f = 0.80	x3, x2	
x5 = (2,2,3)	f = 2.50	x6, x5	
x6 = (3,4,5)	f = 0.50	x6, x4	
x7 = (3,5,7)	f = 0.66	x8, x3	
x8 = (6,8,4)	f = 0.80	x8, x1	



Binary tournament selection

1) Randomly choose pairs of test cases (solutions)
2) Select the fittest (better) individuals from each pair

Tournament Selection

		Tournaments	Winners
x1 = (2,2,3)	f = 2.50	**x2**, x7	x2 = (2,3,5)
x2 = (2,3,5)	f = 0.66	**x1**, x5	X1 = (2,2,3)
x3 = (-2,3,6)	f = 3.66	x3, **x8**	x8 = (6,8,4)
x4 = (2,3,7)	f = 0.80	x3, **x2**	x2 = (2,3,5)
x5 = (2,2,3)	f = 2.50	**x6**, x5	x6 = (3,4,5)
x6 = (3,4,5)	f = 0.50	**x6**, x4	x6 = (3,4,5)
x7 = (3,5,7)	f = 0.66	**x8**, x3	x8 = (6,8,4)
x8 = (6,8,4)	f = 0.80	**x8**, x1	x8 = (6,8,4)

Binary tournament selection

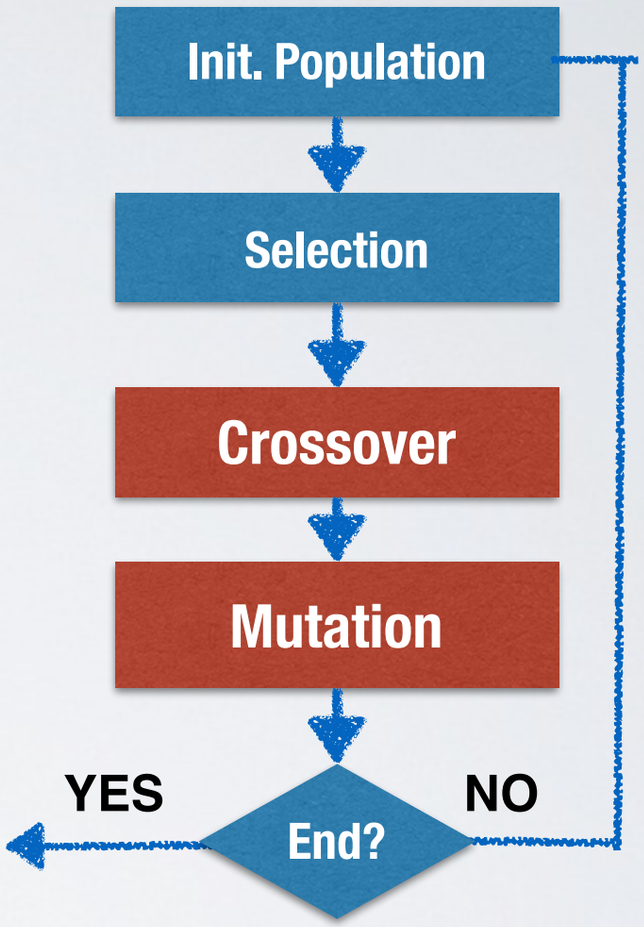
1) Randomly choose pairs of test cases (solutions)
2) Select the fittest (better) individuals from each pair

Reproduction (Crossover)

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
     }// if isTriangle()
  }}
```

Target



Reproduction (Crossover)

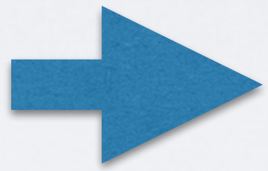
Winners	Parents	Cut-point	Offsprings
x1 = (2,3,5)	x1	-	o1 = (2,3,5)
x2 = (2,2,3)	x2, x5	1	o2 = (2,4,5)
x3 = (6,8,4)	x3, x8	1	o3 = (3,2,3)
x4 = (2,3,5)	x4, x6	2	o4 = (6,8,4)
x5 = (3,4,5)	x7	-	o5 = (6,8,4)
x6 = (3,4,5)			o6 = (2,4,4)
x7 = (6,8,4)			o7 = (6,8,5)
x8 = (6,8,4)			o8 = (6,8,4)

One-point crossover (probability = 0.8)
It takes two parents and cuts their chromosome strings at some randomly chosen position and the produced substrings are then swapped to produce two new full-length chromosomes.

Reproduction (Mutation)

Offsprings

o1 = (2,3,5)
o2 = (2,4,5)
o3 = (3,2,3)
o4 = (6,8,4)
o5 = (6,8,4)
o6 = (2,4,4)
o7 = (6,8,5)
o8 = (6,8,4)



Mutated Offsprings

o1 = (2,3,5)
o2 = (2,4,2)
o3 = (3,2,3)
o4 = (1,8,4)
o5 = (6,8,4)
o6 = (2,4,4)
o7 = (6,5,5)
o8 = (6,8,4)

Mutation: randomly changes some genes (elements within each chromosome)

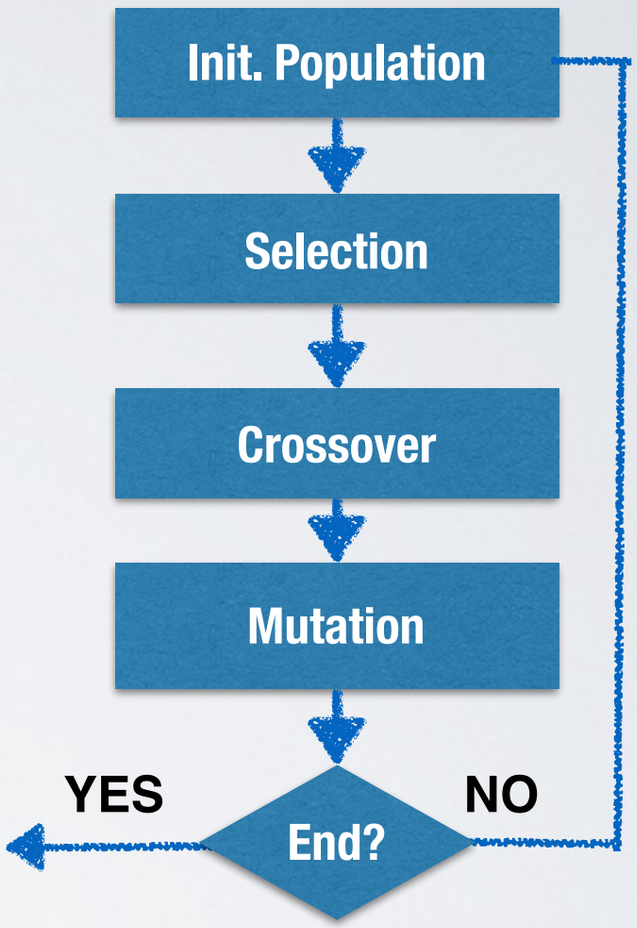
Mutation probability: 1/n where n=chromosome length

Iterating (generations)

TU Delft

```
class Triangle {

 void computeTriangleType() {
1. if (isTriangle()){
2.   if (side1 == side2) {
3.       if (side2 == side3)
4.           type = "EQUILATERAL";
         else
5.           type = "ISOSCELES";
     } else {
6.       if (side1 == side3) {
7.           type = "ISOSCELES";
         } else {
8.           if (side2 == side3)
9.               type = “ISOSCELES”;
             else
10.              checkRightAngle();
             }
         }
     }// if isTriangle()
  }}
```

Target



Running Example

```
class Triangle {
    private double side1, side2, side3;
    private String type = “NOT_A_TRIANGLE”;

    public Triangle (double a, double b, double c){…}
    private void checkRightAngle() {…}
    public void computeTriangleType() {…}
    private boolean isTriangle() {…}
}
```

Running Example

```
class Triangle {
   int a, b, c; //sides
   int type = NOT_A_TRIANGLE;

   Triangle (int a, int b, int c){…}
   void checkRightAngle() {…}
   void computeTriangleType() {…}
   boolean isTriangle() {…}
   public static void main (String args[]) {…}
}
```

Class under test

Initial Test Cases

```
@Test
public void test(){
   Triangle c = new Triangle(8.0, 2.1
   c.computeTriangleType();
   assertTrue
}
```

```
@Test
public void test(){
   Triangle c = new Triangle(1
   c.computeTriangleType();
   assertTrue(c.getTriangleTyp
}
```

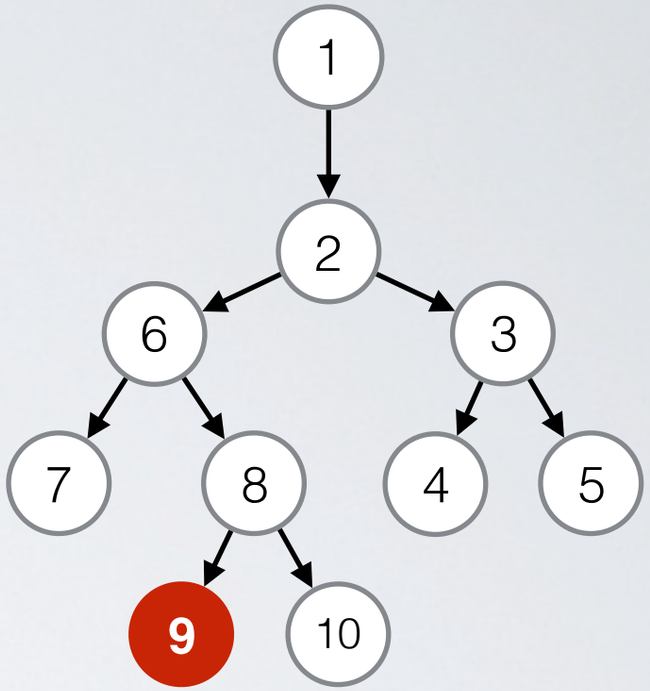
```
@Test
public void test(){
   Triangle c = new Triangle(1.0, 2.1, 4.2);
   c.computeTriangleType();
   assertTrue(c.getTriangleType, …);
}
```

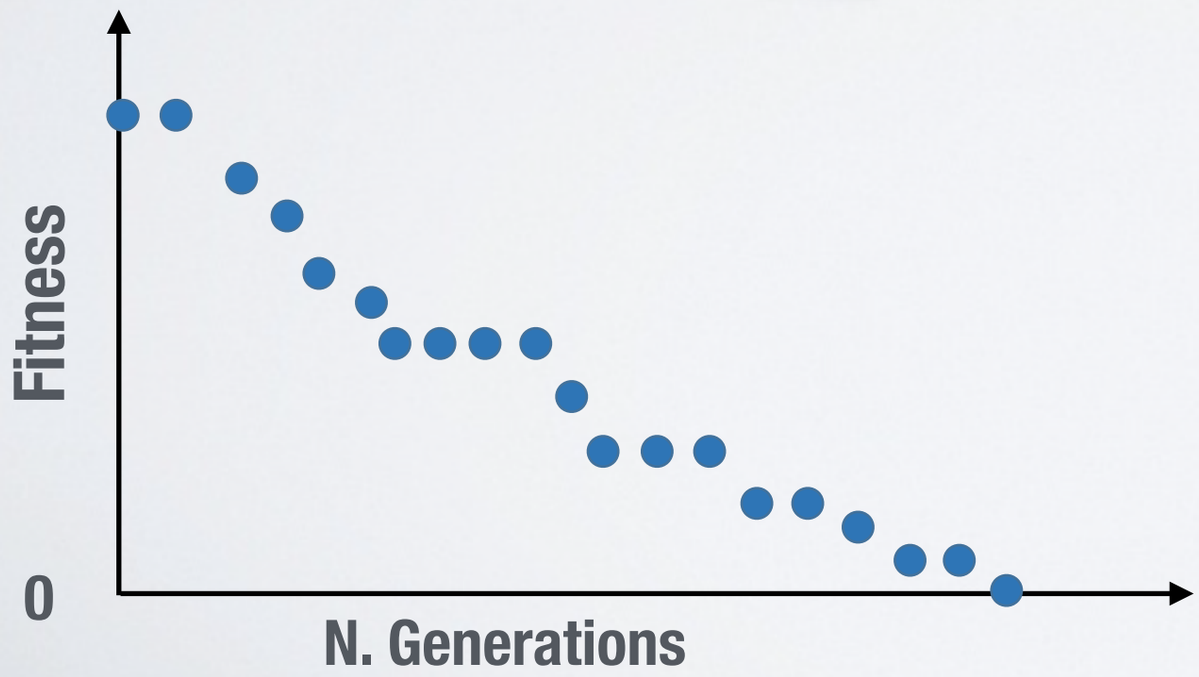
```
@Test
public void test(){
   Triangle c = new Triangle(-2.0, 12.0, 0);
   c.computeTriangleType();
   assertTrue(c.getTriangleType, …);
}
```

Running Example

```
class Triangle {

 void computeTriangleType() {
   if (isTriangle()){
     if (side1 == side2) {
          if (side2 == side3)
              type = "EQUILATERAL";
          else
              type = "ISOSCELES";
      } else {
          if (side1 == side3) {
              type = "ISOSCELES";
           } else {
             if (side2 == side3)
                 type = “ISOSCELES”;
             else
                checkRightAngle();
                }
          }
     }// if isTriangle()
}}
```

Target

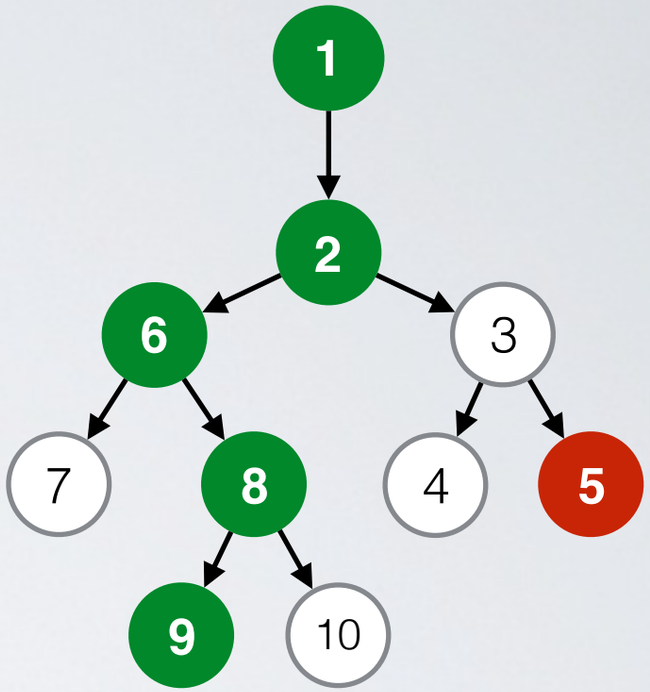


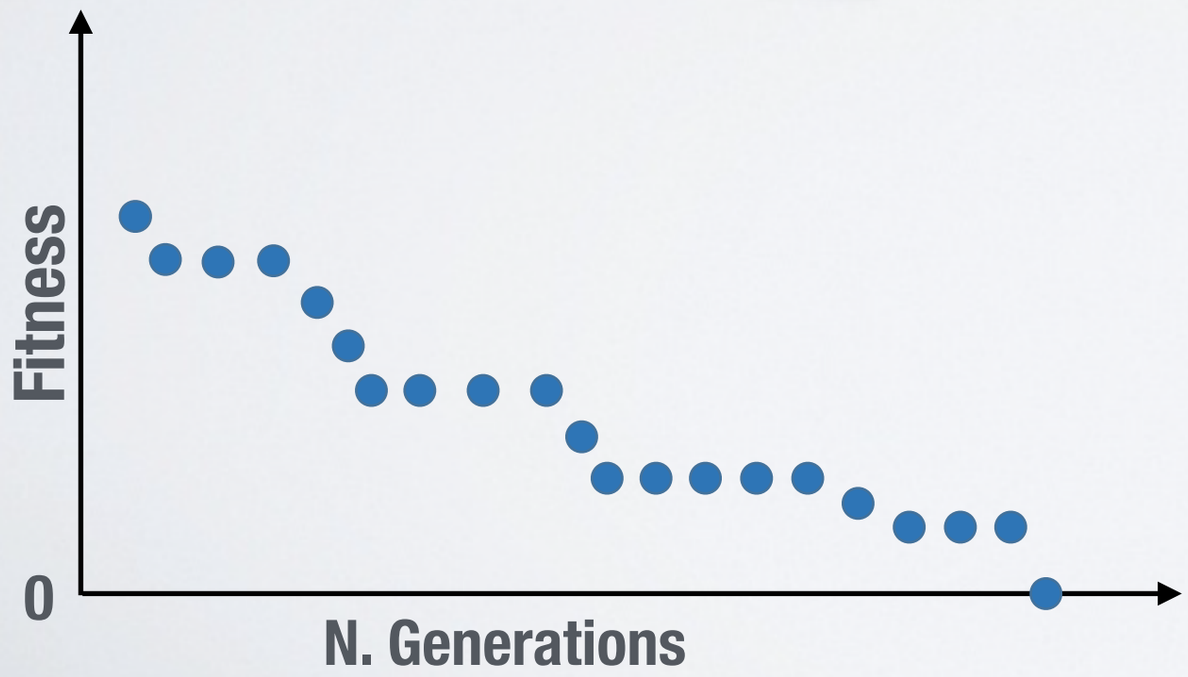


Running Example

```
class Triangle {

 void computeTriangleType() {
   if (isTriangle()){
     if (side1 == side2) {
          if (side2 == side3)
              type = "EQUILATERAL";
          else
              type = "ISOSCELES";
      } else {
          if (side1 == side3) {
              type = "ISOSCELES";
           } else {
             if (side2 == side3)
                 type = “ISOSCELES”;
             else
                checkRightAngle();
                }
          }
      }// if isTriangle()
}}
```

Target



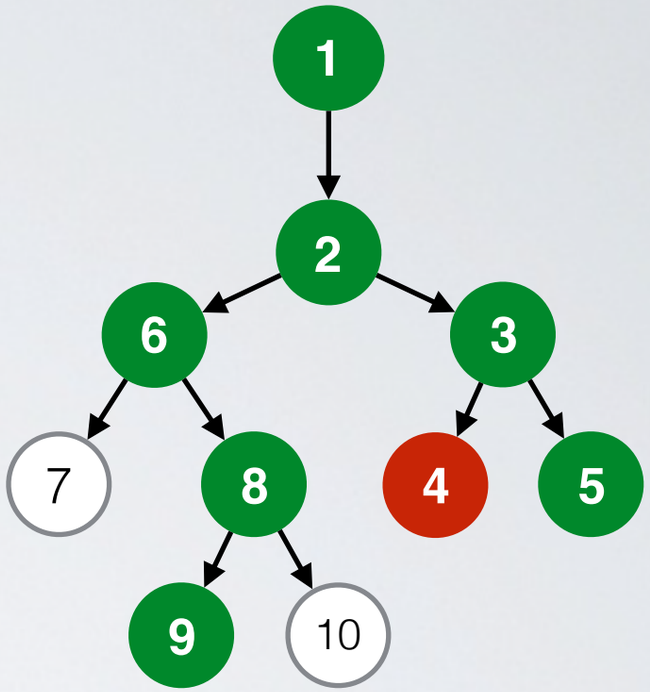


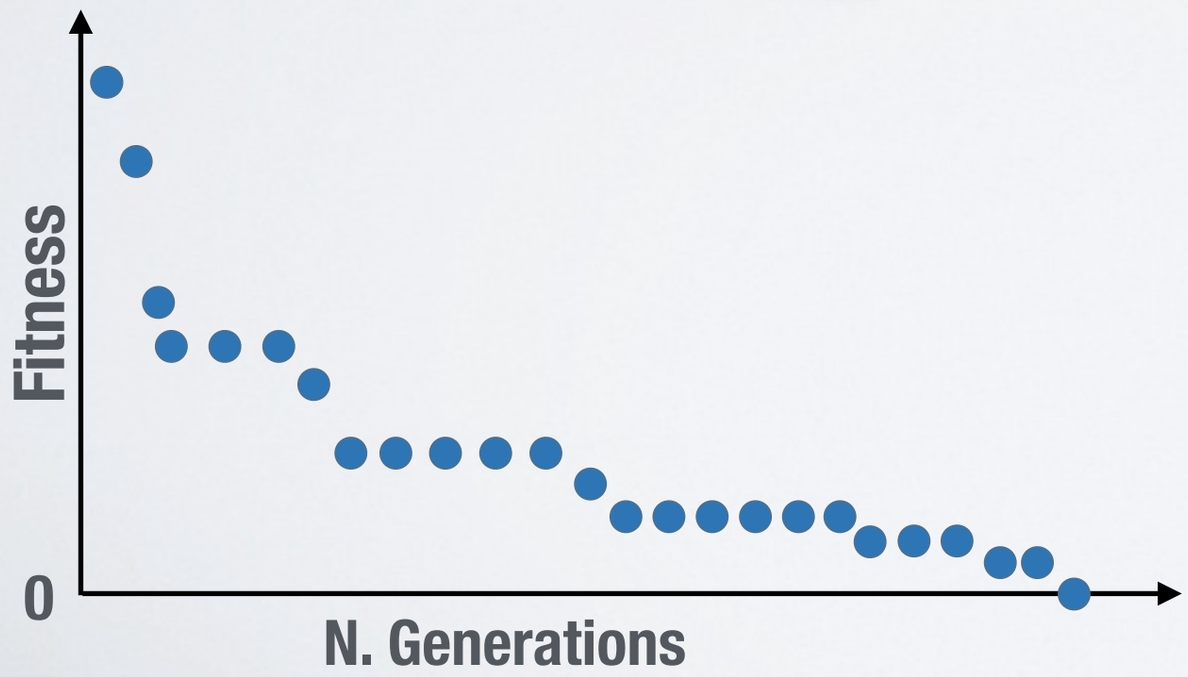
TC1 = (4,3,3)

Running Example

```
class Triangle {

 void computeTriangleType() {
   if (isTriangle()){
     if (side1 == side2) {
          if (side2 == side3)
              type = "EQUILATERAL";
          else
              type = "ISOSCELES";
      } else {
          if (side1 == side3) {
              type = "ISOSCELES";
           } else {
             if (side2 == side3)
                 type = “ISOSCELES”;
             else
                checkRightAngle();
                }
          }
     }// if isTriangle()
}}
```

Target



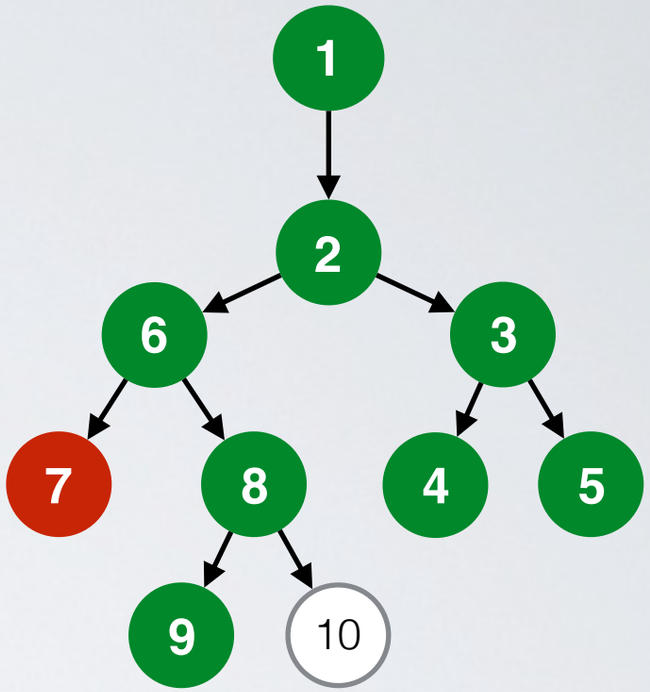


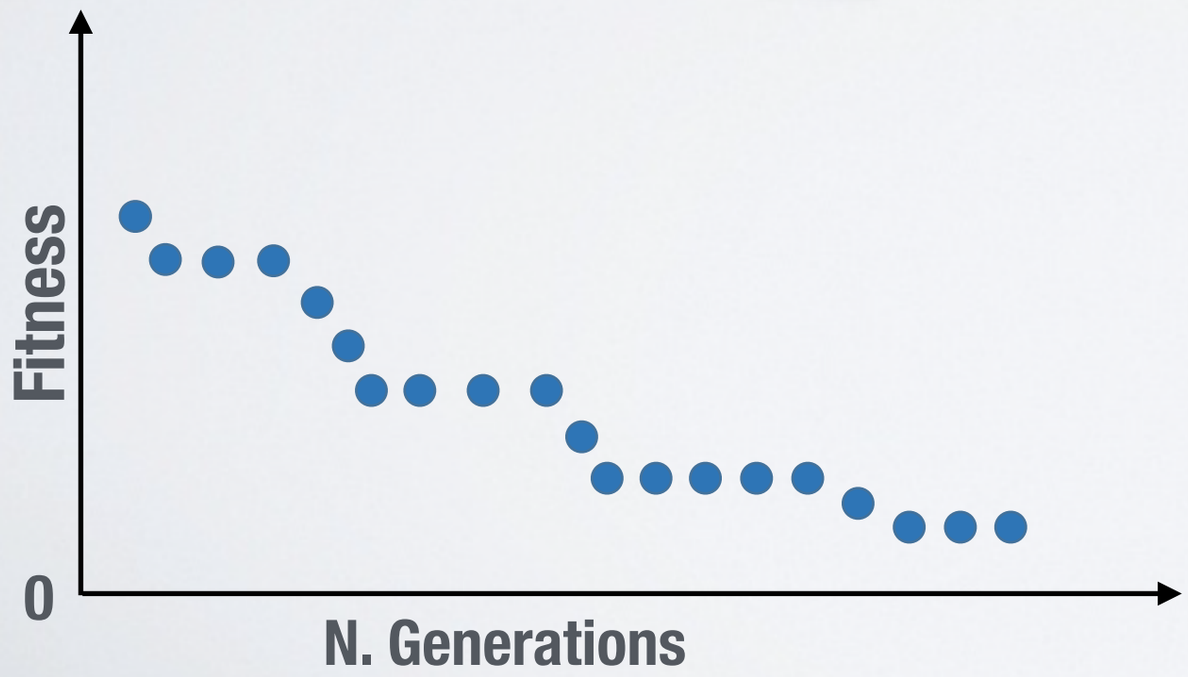
TC1 = (4,3,3)
TC2 = (2,2,4)

Running example

```
class Triangle {

 void computeTriangleType() {
    if (isTriangle()){
      if (side1 == side2) {
           if (side2 == side3)
               type = "EQUILATERAL";
           else
               type = "ISOSCELES";
       } else {
           if (side1 == side3) {
               type = "ISOSCELES";
            } else {
              if (side2 == side3)
                  type = “ISOSCELES”;
              else
                 checkRightAngle();
                 }
           }
      }// if isTriangle()
 }}
```

Target

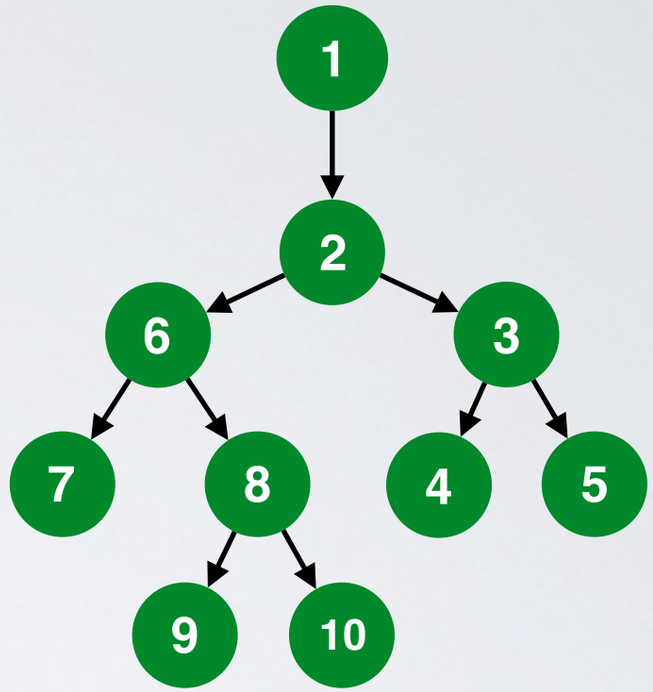




TC1 = (4,3,3)
TC2 = (2,2,4)
TC2 = (5,5,5)

Running example

```
class Triangle {

 void computeTriangleType() {
   if (isTriangle()){
     if (side1 == side2) {
          if (side2 == side3)
              type = "EQUILATERAL";
          else
              type = "ISOSCELES";
      } else {
          if (side1 == side3) {
              type = "ISOSCELES";
           } else {
             if (side2 == side3)
                 type = “ISOSCELES”;
             else
                checkRightAngle();
                }
         }
     }// if isTriangle()
}}
```




```
TC1 = (4,3,3)
TC2 = (2,2,4)
TC3 = (5,5,5)
TC4 = (4,3,4)
TC5 = (1,2,3)
```

The final test suite consists of all chromosome that have been found to cover (even accidentally) one or more yet to cover statements.

Tools

EVOSUITE

http://www.evosuite.org

- Command Line
- Eclipse Plugin
- Intellij IDEA plugin
- Maven Plugin
- Measure Code Coverage

EVOSUITE

Automatic Test Suite Generation for Java

HOME CONTACT ABOUT DOCUMENTATION PUBLICATIONS EXPERI

New 1.0.6 release

A new version 1.0.6 of EvoSuite has now been released, and again contains a bunch of bug fixes. Many changes relate to the mocking infrastructure/Mockito integration as well as general crashes, and the search algorithms have seen some refactoring. Release data is available 1.0.6 release results.

Search

RECENT POSTS

New 1.0.6 release

EvoSuite wins the SBST 2017 tool compe-

https://github.com/EvoSuite/evosuite

EvoSuite / evosuite

Used by 1 | Unwatch 31 | Unstar 237 | Fork 133

Code | Issues 63 | Pull requests 3 | Projects 0 | Wiki | Security | Insights

EvoSuite - automated generation of JUnit test suites for Java classes http://www.evosuite.org

7,859 commits | 10 branches | 5 releases | 16 contributors | View license

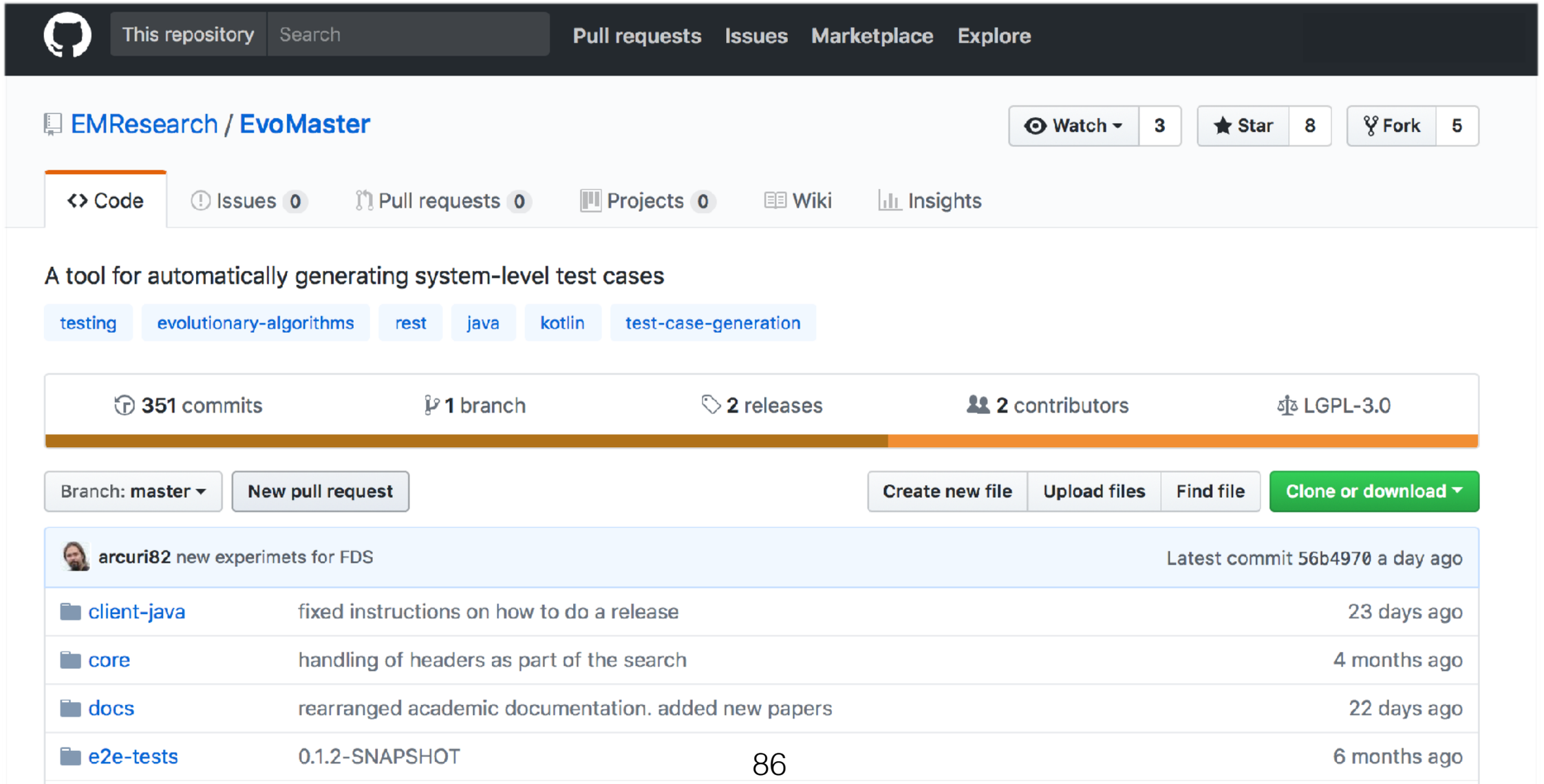
Branch: master | New pull request | Create new file | Upload files | Find File | Clone or download

apanichella Merge branch 'apanichella-archive-dynamosa' — Latest commit 1895b6d on Feb 14

.github	Simplify issue template	2 years ago
client	Merge branch 'archive-dynamosa' of https://github.com/apanichella/evo...	3 months ago
generated	Update version number post release	a year ago
master	Merge branch 'master' of https://github.com/EvoSuite/evosuite	6 months ago
plugins	Temporary fix related to issue #233. And not showing a dialog if usin...	6 months ago
release_results	Release result scripts	a year ago
removed	Updated license header.	a year ago
runtime	Minimised diff BaderV vs master branch.	7 months ago
shaded	Update version number post release	a year ago

EvoMaster (System Testing)

https://github.com/EMResearch/EvoMaster



Botsing (Crash Replication)

https://github.com/STAMP-project/botsing

STAMP-project / **botsing**

Watch 11 | Star 11 | Fork 5

Code | Issues 13 | Pull requests 0 | Projects 0 | Insights

Botsing is a Java framework for crash reproduction. It depends on EvoSuite (http://www.evosuite.org) for code instrumentation.

h2020 | stamp | java-framework | crash-reproduction | java

329 commits | **2** branches | **4** releases | **8** contributors | Apache-2.0

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

pderakhshanfar Add the building instruction to the README — Latest commit `fb6e611` 11 days ago

.github	Adding issue templates	6 months ago
botsing-examples	[maven-release-plugin] prepare for next development iteration	a month ago
botsing-maven	[maven-release-plugin] prepare for next development iteration	a month ago
botsing-model-generation	[maven-release-plugin] prepare for next development iteration	a month ago
botsing-parsers	Removing trailing space	19 days ago
botsing-preprocessing	[maven-release-plugin] prepare for next development iteration	a month ago
botsing-reproduction	Fix classpath option in help message and README	25 days ago
docs	Add Botsing demo note	a month ago
.gitignore	ignoring generated dependency-reduced-pom.xml file	a month ago

The Sapienz Project at Facebook

https://arstechnica.com/information-technology/2017/08/facebook-dynamic-analysis-software-sapienz/

BIZ & IT —

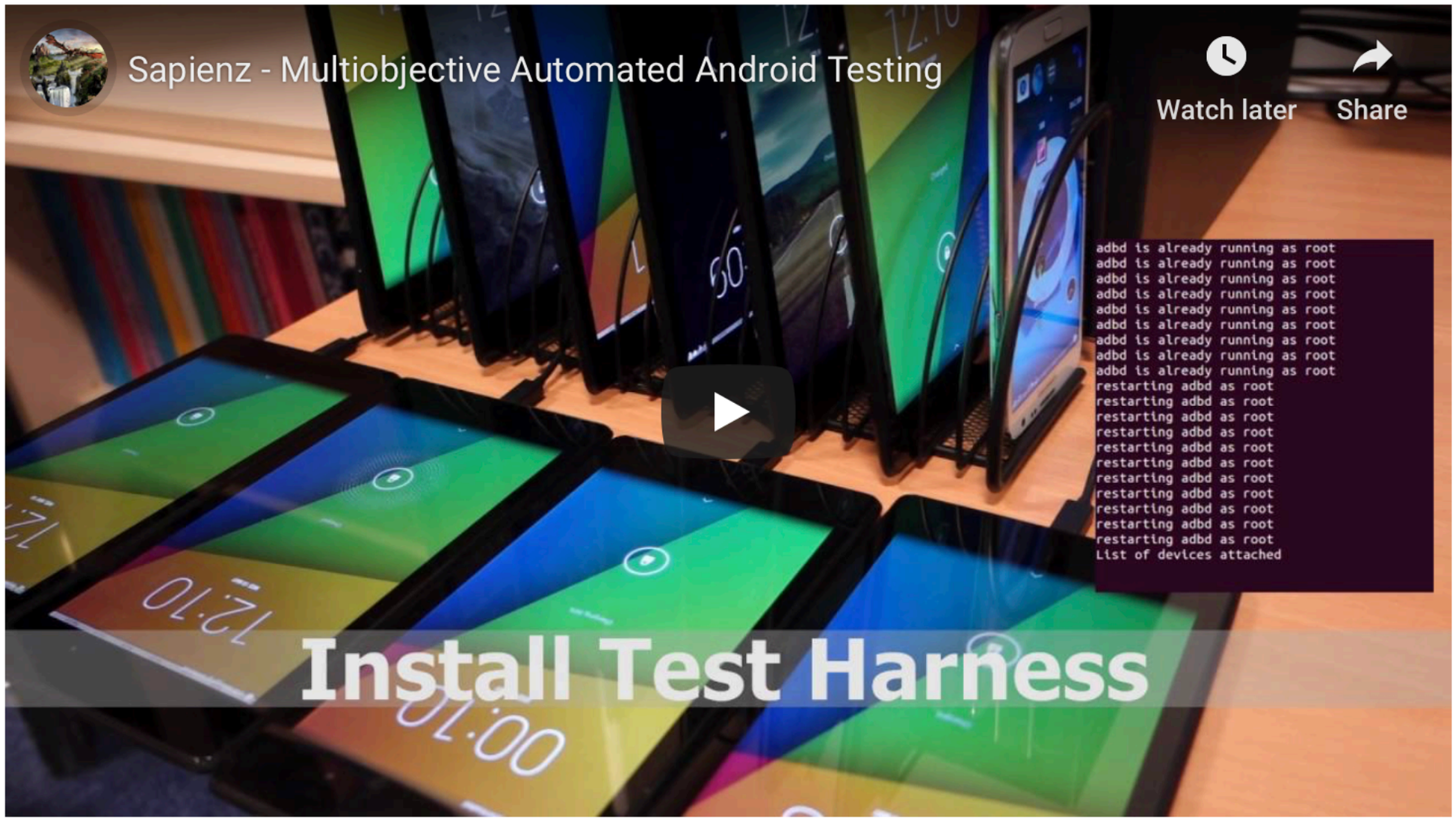
Facebook's evolutionary search for crashing software bugs

Ars gets the first look at Facebook's fancy new dynamic analysis tool.

SEBASTIAN ANTHONY - 8/22/2017, 4:26 PM

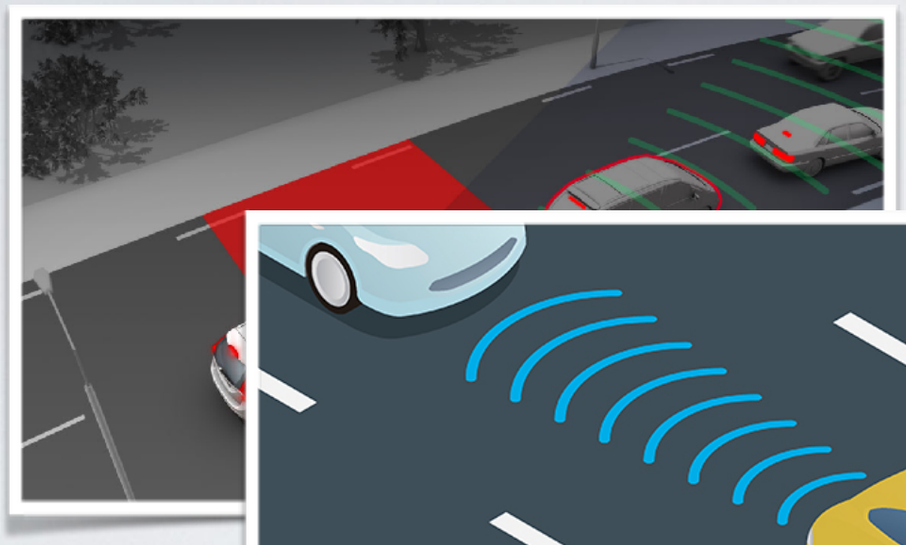
Sapienz

Sapienz in action: https://youtu.be/j3eV8NiWLg4

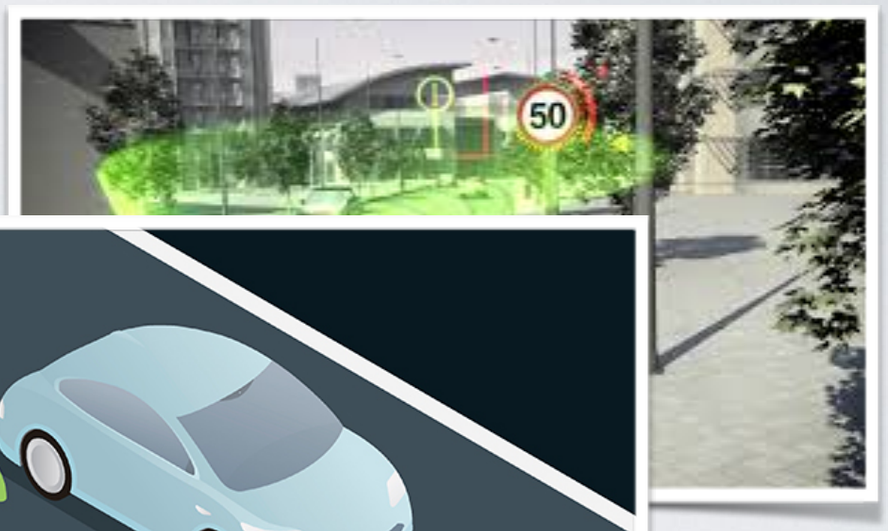


Case Study: Testing Self-driving Cars with GA

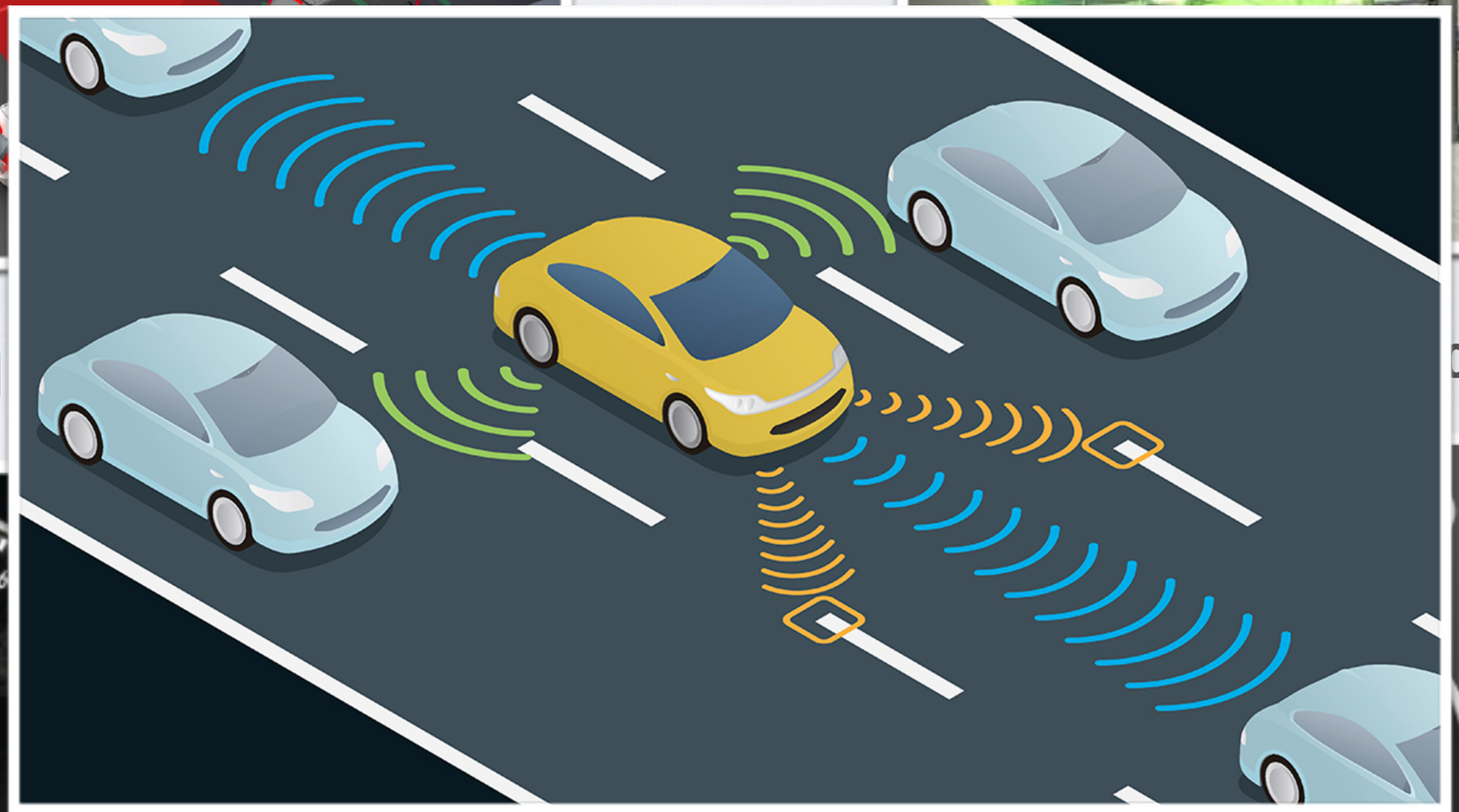
Advanced Driver Assistance Systems (ADAS)



Automated



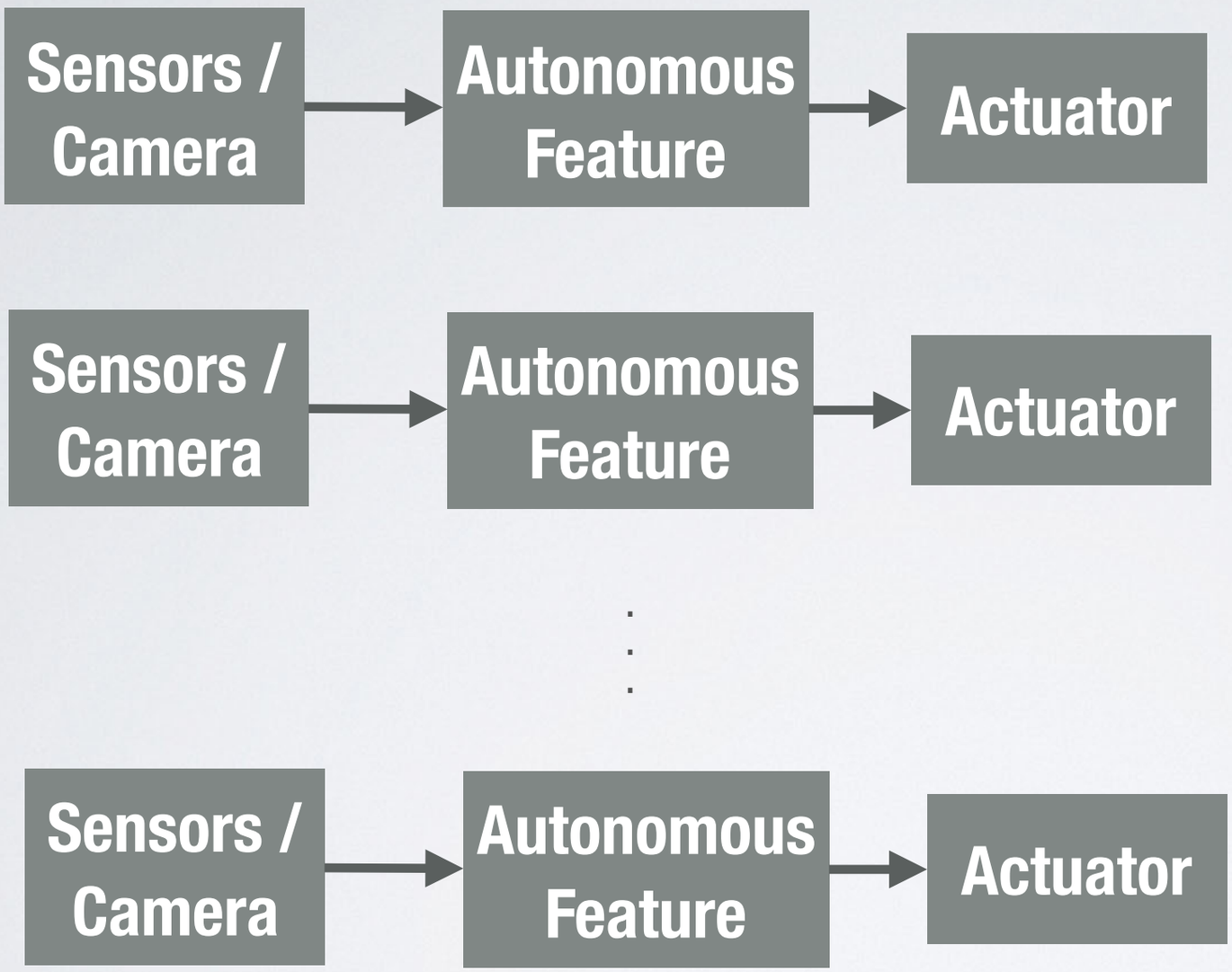
on (TSR)

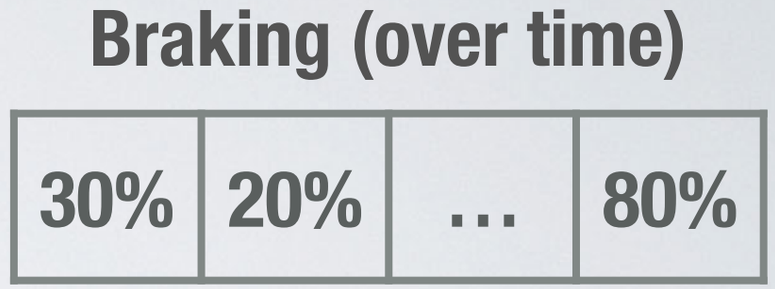


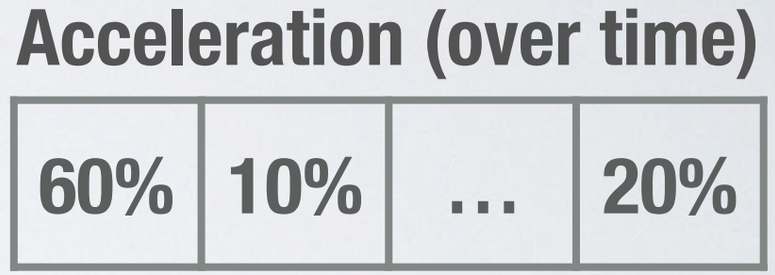
Pedestrian Protection (PP)

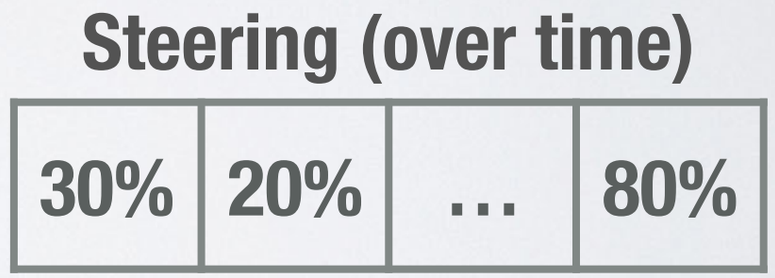
Lane Departure Warning (LDW)

Feature Interactions

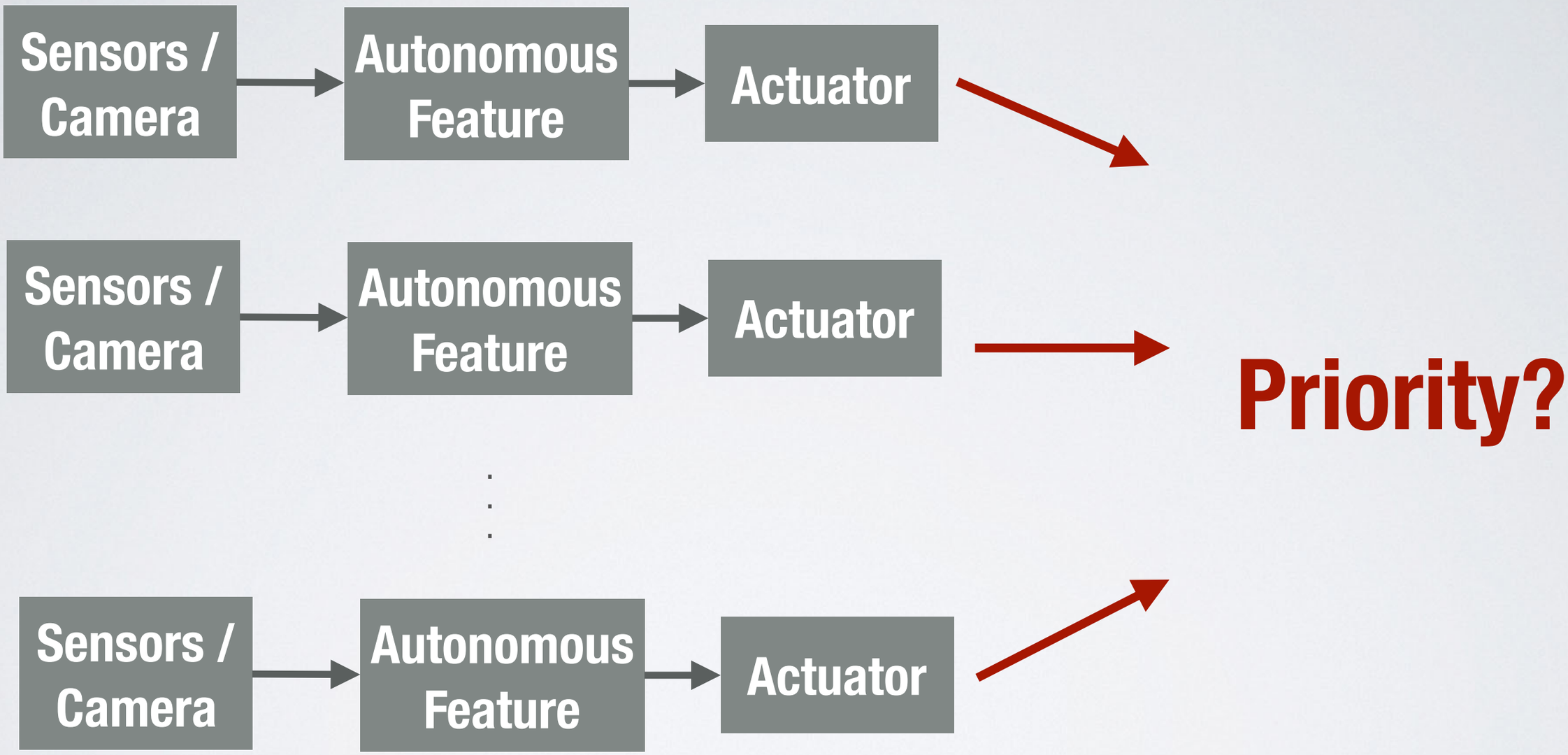




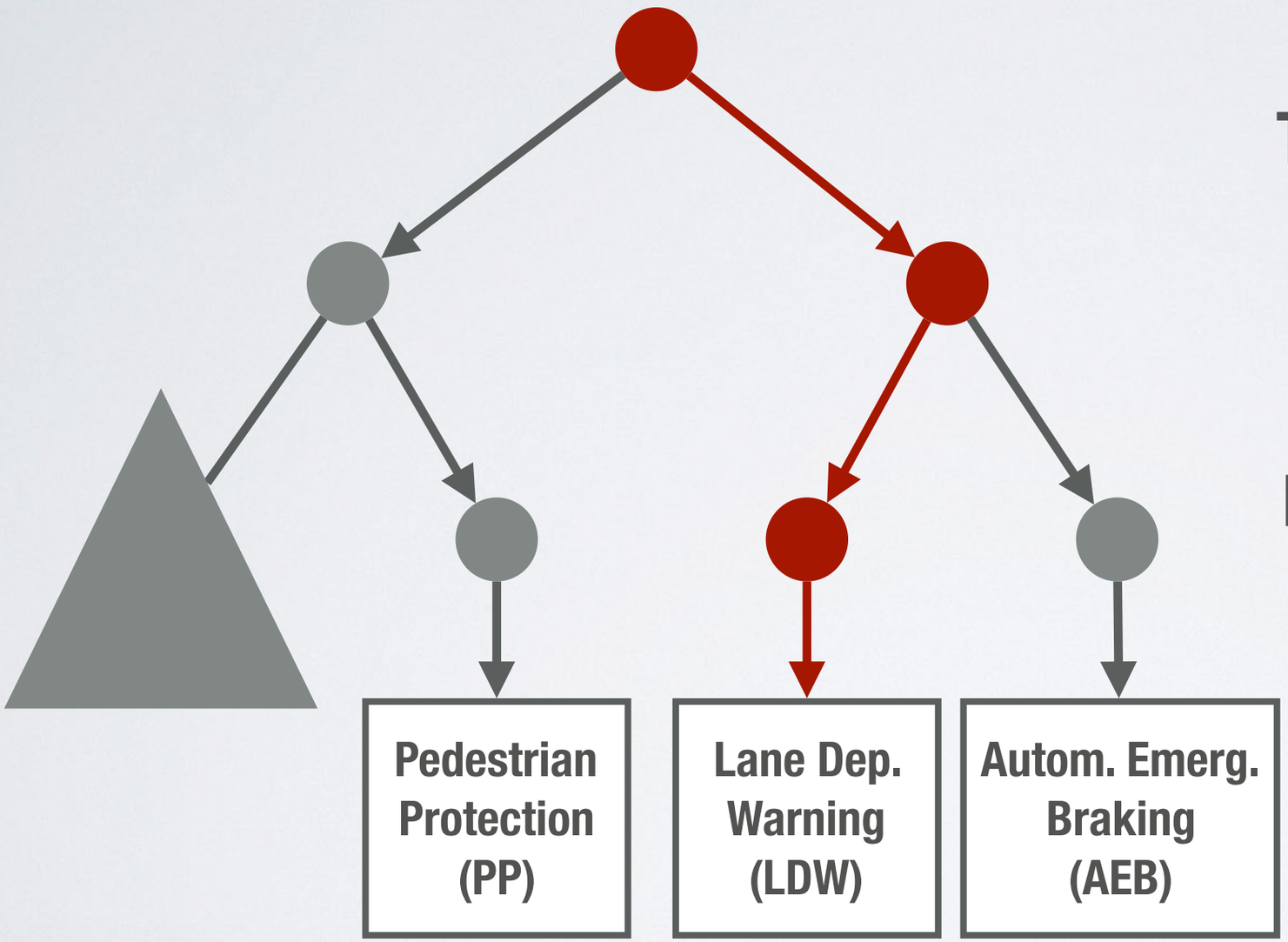




Feature Interactions



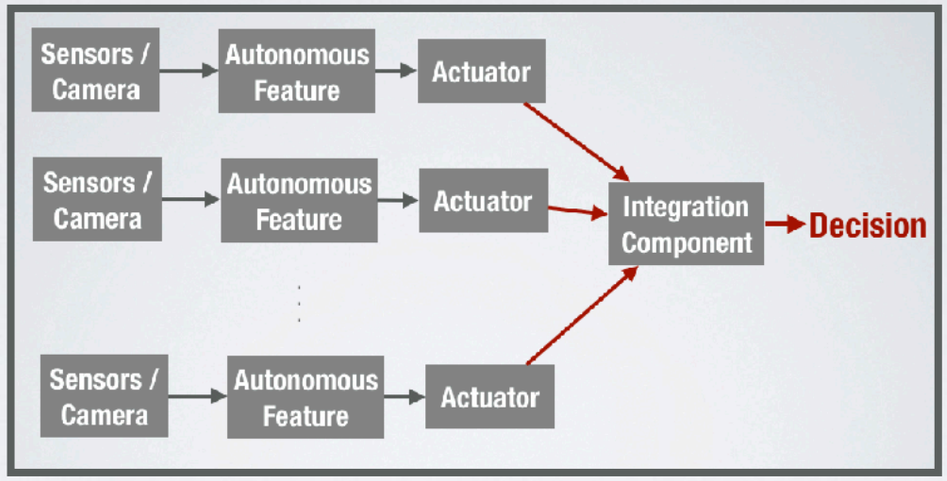
Integration Components



The integration is a rule set: each condition checks a specific feature interaction situation and resolves potential conflicts that may arise under that condition

Testing Using Physics-Based Simulation

Test Input



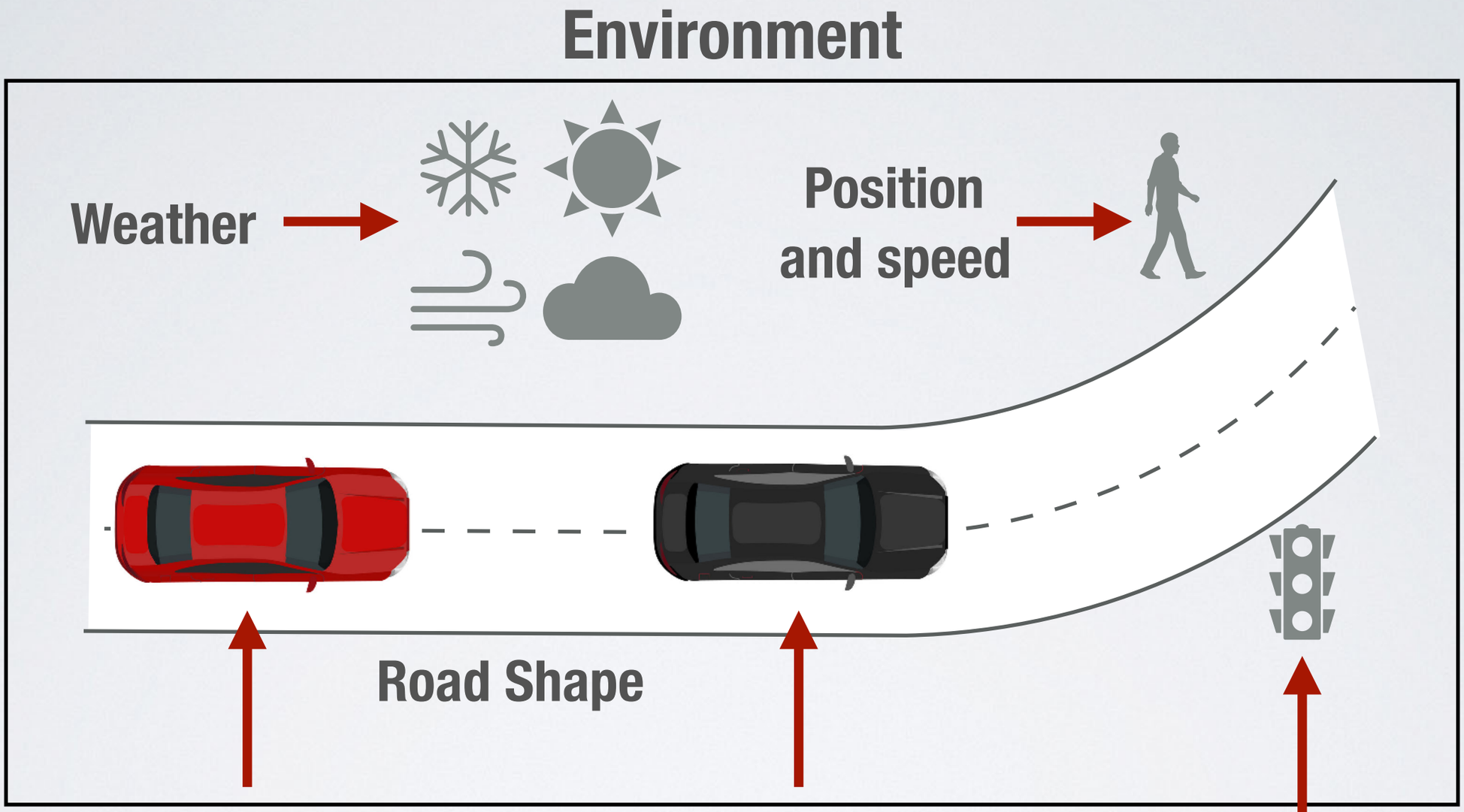
Software Under Test (SUT)

Simulator (Matlab/Simulink)

- Physical plant (vehicle / sensors / actuators)
- Other cars
- Pedestrians
- Environment (weather / roads / traffic signs)

Test Output

Test Inputs

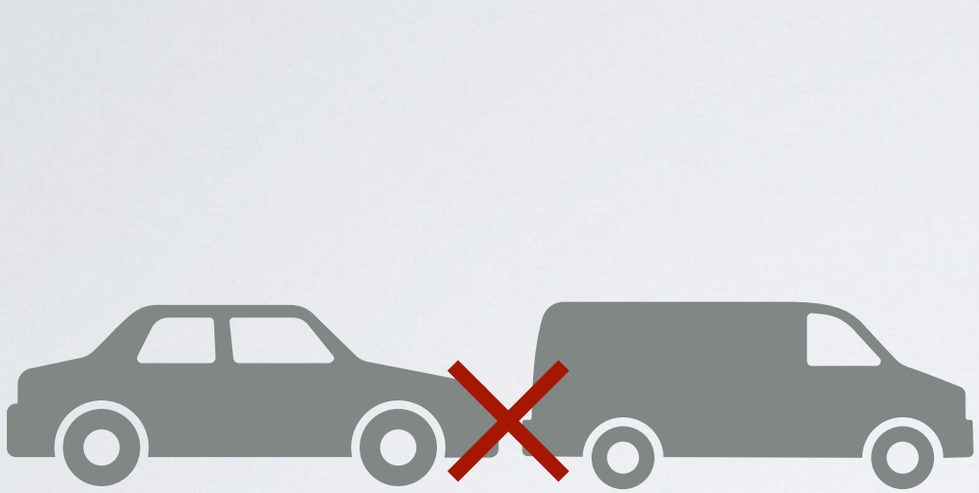


Car Under Test:
- Initial Position
- Initial Speed

Ego Car:
- Initial Position
- Initial Speed

Traffic lights position and status

Testing Target: Feature Interactions Failures

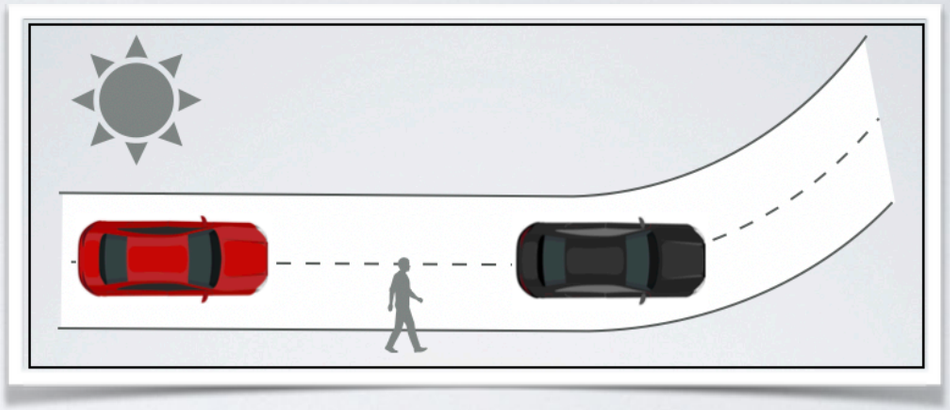


Infinite Test Space

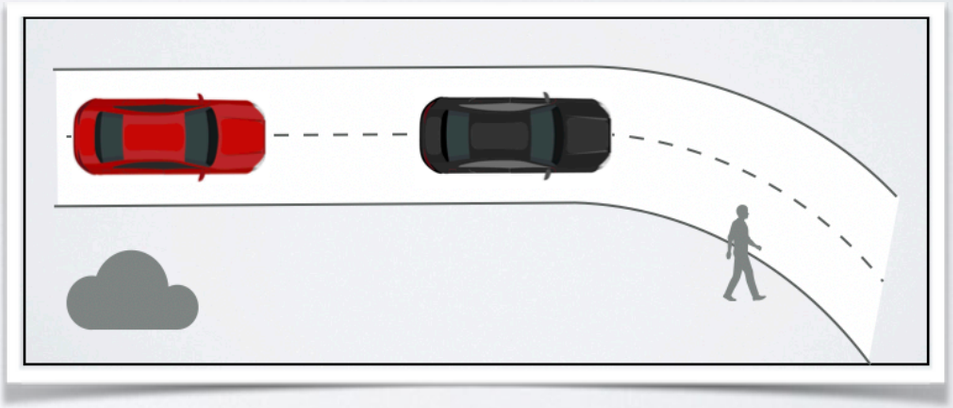
AI-Based Testing

AI-Based Testing

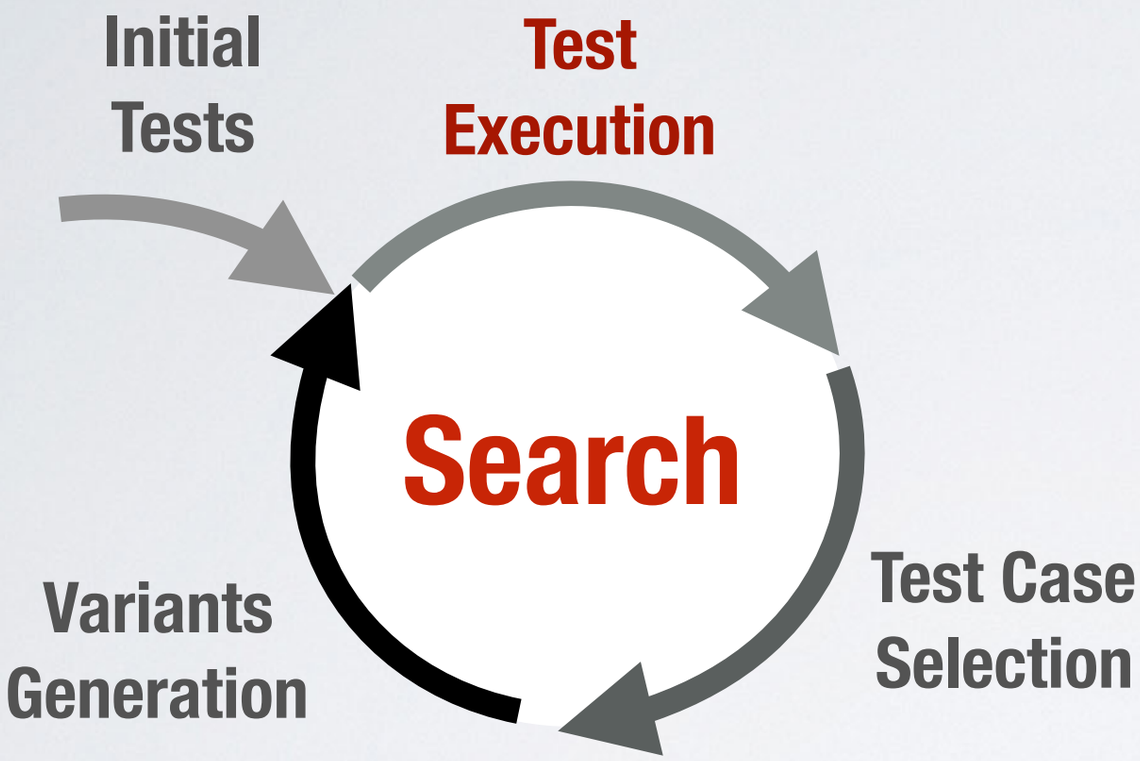
Test 1



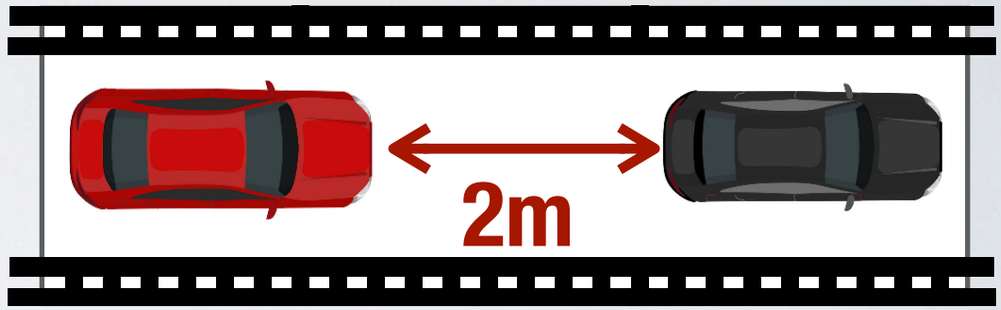
Test 2



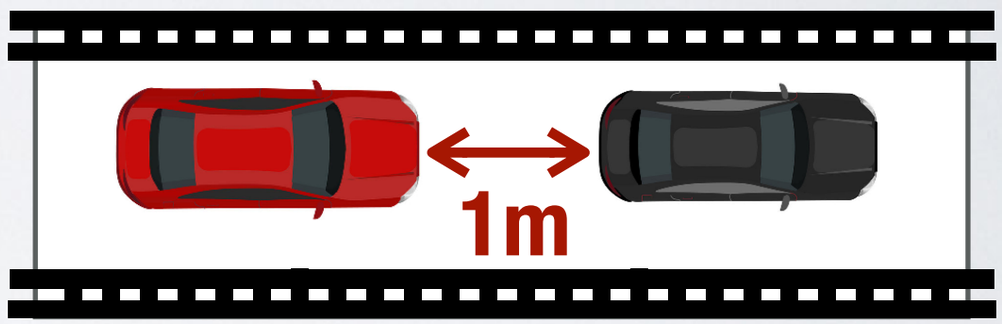
AI-Based Testing



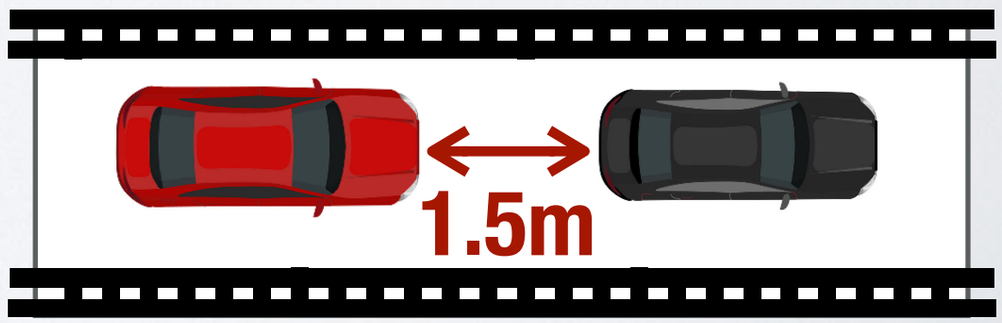
Results of Test 1



Results of Test 2

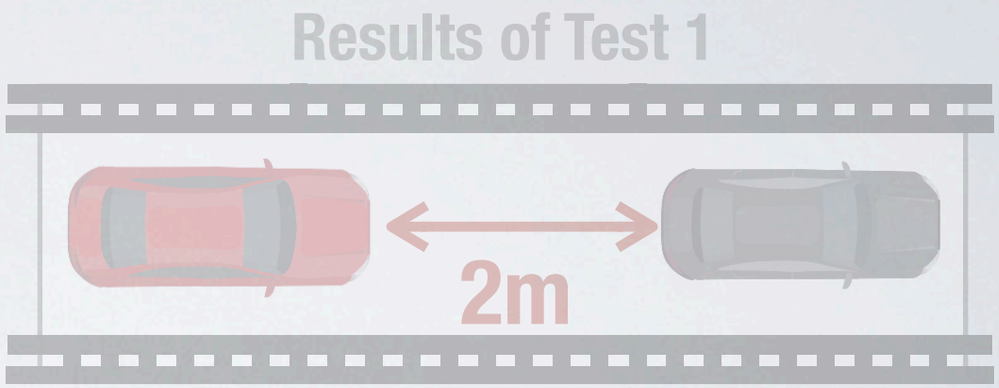


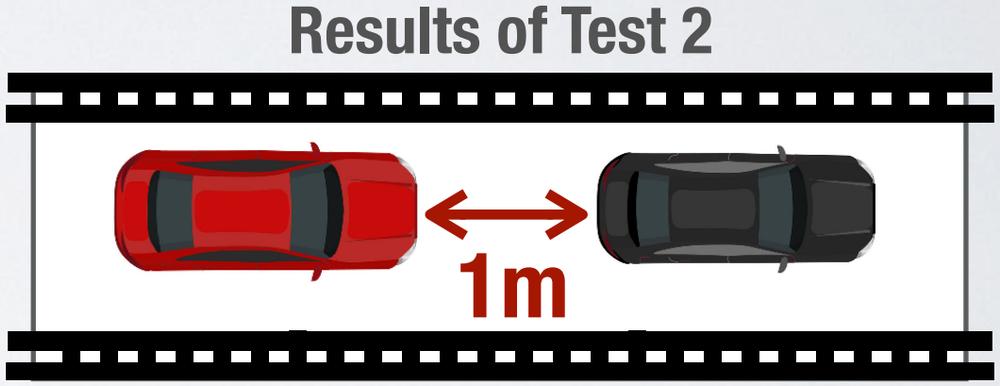
Results of Test 3



Minimum distance within the simulation time window

AI-Based Testing

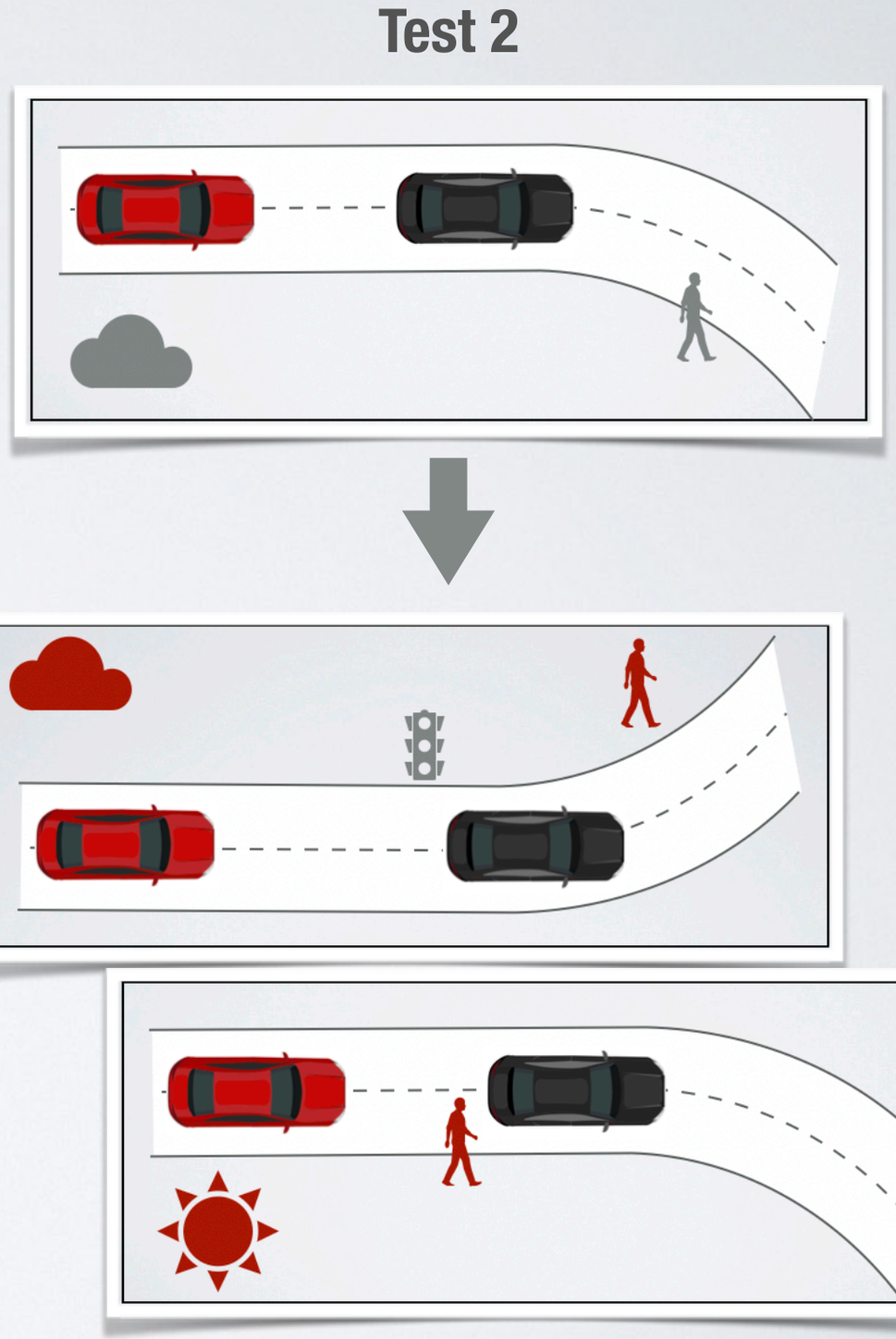




The best test case it the one **closer** to **violate** the safe **distance** (fitness)

AI-Based Testing

Mutation and/or Crossover

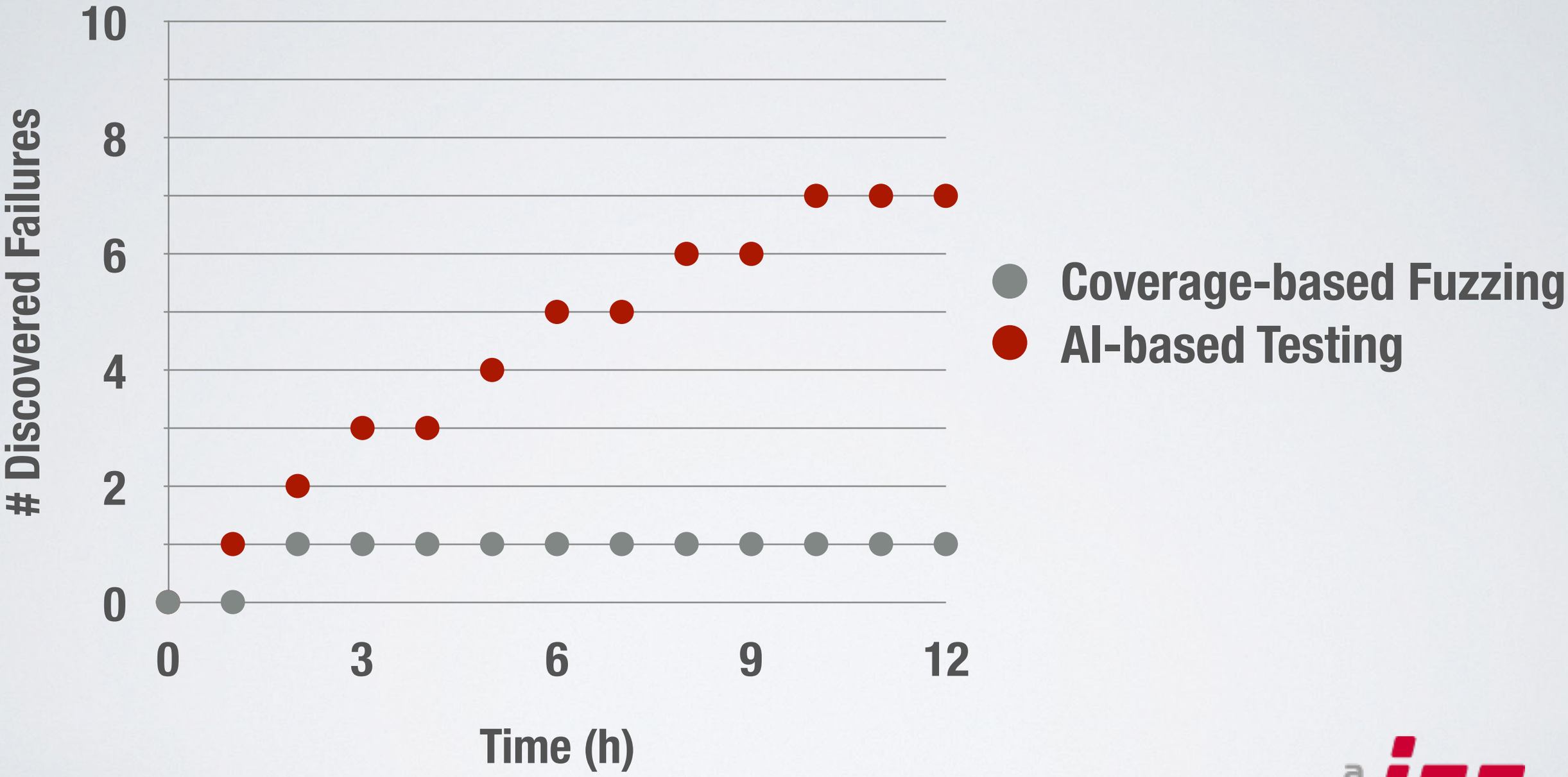


Case Study

- Two case study systems from IEE (industrial partner)
  - Designed by experts
  - Manually tested for more than six months
  - Different rules to integrated feature actuator commands

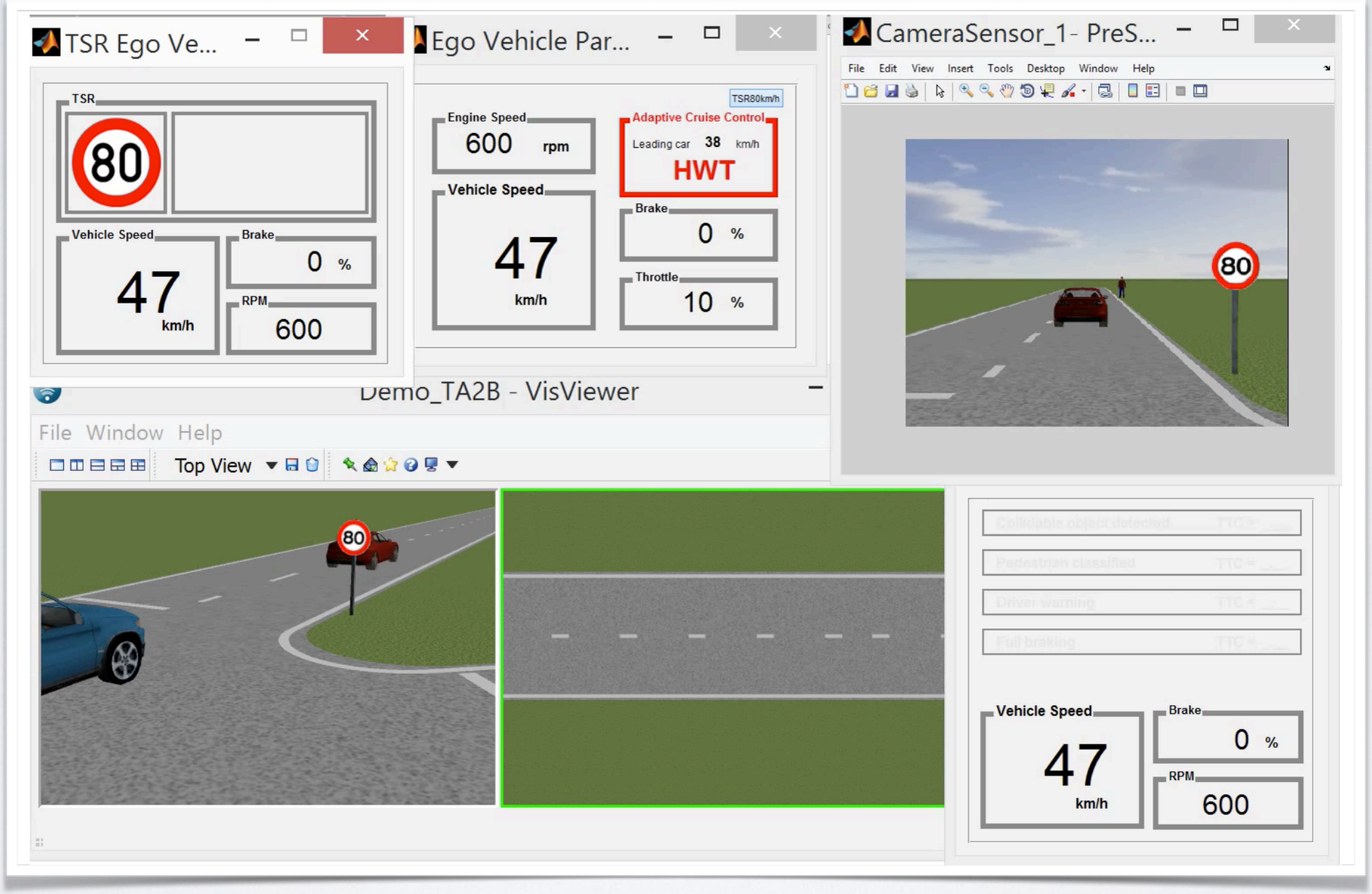
- Both systems consist of four self-driving features
  - Adaptive Cruise Control (ACC)
  - Automated Emergency Braking (AEB)
  - Traffic Sign Recognition (TSR)
  - Pedestrian Protection (PP)

Some Results



a sense for innovation IEE

Example of Failures



Summary

- White-box Unit Testing
- Random Testing (Fuzzing)
- Search-based Software Testing
- Genetic Algorithms
- Test Case Generation
- Tools