Search-Based Software Engineers Need Tools

Gordon Fraser, University of Sheffield
Contents

1. What is Search Based Software Testing?
2. Building an SBST Tool is Easy!
3. The EvoSuite Test Generation Tool
4. Lessons Learned Building an SBST Tool
Automated test generation
package example;

public class Foo {
    private int x = 0;
    private String str;
    private String str2 = "bar";

    public Foo(String string) {
        this.str = string;
    }

    public void inc() {
        x++;
    }

    public boolean coverMe() {
        if (x == 5)
            if (!str.equals(str2))
                if (str.equalsIgnoreCase(str2))
                    return true;

        return false;
    }
}
```java
// Regression assertion (captures the current behavior of the program)
assertTrue(var2 == false);
```
Random Test Data Generation
The code is a puzzle. Do you understand what the code does? Click Ask Pex! to find out.

```csharp
using System;

public class Foo {
    private int x = 0;
    private String str;
    private String str2="bar";
    public Foo(String str) {
        this.str = str;
    }
    public void inc() {
        x++;
    }
    public bool coverMe() {
        if (x==5)
            if (!str.Equals(str2))
                if (str.Equals(str2))
```
The code is a puzzle. Do you understand what the code does? Click Ask Pex! to find out.

```csharp
public class Example
{
    public string str;

    public void inc()
    {
        x++;  
    }

    public bool coverMe()
    {
        if (x==5)
        {
            if (!str.Equals(str2))
            {
                if (str.Equals(str2, StringComparison.OrdinalIgnoreCase))
                    return true;
            }
        }
        return false;
    }
}
```
```csharp
public bool coverMe() {
    if (x==5)
        if (!str.Equals(str2))
            if (str.Equals(str2,
                StringComparison.OrdinalIgnoreCase))
                return true;
    return false;
}

public static bool Puzzle(Foo foo) {
    return foo.coverMe();
}
```

### Output/Exception Table

<table>
<thead>
<tr>
<th>foo</th>
<th>result</th>
<th>Output/Exception</th>
<th>Error Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td></td>
<td>NullReferenceException</td>
<td>Object reference not set to an instance of an object.</td>
</tr>
<tr>
<td>new Foo{}</td>
<td>false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generating vs Checking

Conventional Software Testing Research

Write a method to construct test cases

Search-Based Testing

Write a method to determine how good a test case is
Generating vs Checking

Conventional Software Testing Research
Write a method to construct test cases

Search-Based Testing
Write a fitness function to determine how good a test case is
Fitness-guided search
Fitness-guided search

Fitness vs. Input
Components of an SBST Tool

```python
def testMe(x, y):
    if x == 2 * (y + 1):
        return True
    else:
        return False
```

- Search Algorithm
- Representation
- Fitness Function
## Components of an SBST Tool

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search Algorithm</td>
<td>Meta-heuristic algorithm</td>
</tr>
<tr>
<td>Representation</td>
<td>Encoding of the problem solution</td>
</tr>
<tr>
<td>Search Operators</td>
<td>Modifications of encoded solutions</td>
</tr>
<tr>
<td>Fitness Function</td>
<td>Measure how good a candidate solution is</td>
</tr>
<tr>
<td>Test Execution</td>
<td>Execute tests</td>
</tr>
<tr>
<td>Instrumentation</td>
<td>Collect data/traces for fitness calculation during execution</td>
</tr>
</tbody>
</table>
package example;

public class Foo {
    private int x = 0;
    private String str;
    private String str2 = "bar";
    public Foo(String string) {
        this.str = string;
    }
    public void inc() {
        x++;
    }
    public boolean coverMe() {
        if (x == 5) {
            if (!str.equals(str2)) {
                if (str.equalsIgnoreCase(str2)) {
                    return true;
                }
            }
        }
        return false;
    }
}
<table>
<thead>
<tr>
<th>First name</th>
<th>Last name</th>
<th>E-mail</th>
<th>Phone</th>
<th>Mobile</th>
</tr>
</thead>
</table>

Create a category

Category name: eO^L already exists

[Abbrechen] [OK]
Contents

1. What is Search Based Software Testing?
2. Building an SBST Tool is Easy!
3. The EvoSuite Test Generation Tool
4. Lessons Learned Building an SBST Tool
def testMe(x, y):
    if x == 2 * (y + 1):
        return True
    else:
        return False
Components of an SBST Tool

- Search Algorithm: Hill-climbing
- Representation
- Search Operators
- Fitness Function
- Test Execution
- Instrumentation
def testMe(x, y):
    if x == 2 * (y + 1):
        return True
    else:
        return False
Components of an SBST Tool

- Search Algorithm: Hill-climbing
- Representation: Tuple \((x, y)\)
- Search Operators: Neighbourhood of \((x, y)\)
- Fitness Function
- Test Execution
- Instrumentation
Hill Climbing

1. Select Random Value
Hill Climbing

2. Explore Neighbourhood
Hill Climbing

3. Choose better neighbour
Hill Climbing

4. Repeat until optimum is found
Components of an SBST Tool

- Search Algorithm: Hill-climbing
- Representation: Tuple (x, y)
- Search Operators: Neighbourhood of (x, y)
- Fitness Function
- Test Execution
- Instrumentation
Input ➔ SUT ➔ Output
Test Data → Instrumented SUT → Fitness
Input → Output → Trace

def testMe(x, y):
    if x == 2 * (y + 1):
        return True
    else:
        return False
Components of an SBST Tool

- **Search Algorithm**: Hill-climbing
- **Representation**: Tuple \((x, y)\)
- **Search Operators**: Neighbourhood of \((x, y)\)
- **Fitness Function**: Branch distance
- **Test Execution**: Call method
- **Instrumentation**: Global variable
def testMe(x, y):
    if x == 2 * y and y > 1:
        return True
    else:
        return False
## Branch Distance

<table>
<thead>
<tr>
<th>Expression</th>
<th>Distance True</th>
<th>Distance False</th>
</tr>
</thead>
<tbody>
<tr>
<td>x == y</td>
<td>$</td>
<td>x - y</td>
</tr>
<tr>
<td>x != y</td>
<td>1</td>
<td>$</td>
</tr>
<tr>
<td>x &gt; y</td>
<td>$y - x + 1$</td>
<td>x - y</td>
</tr>
<tr>
<td>x &gt;= y</td>
<td>$y - x$</td>
<td>x - y + 1</td>
</tr>
<tr>
<td>x &lt; y</td>
<td>x - y + 1</td>
<td>x - y</td>
</tr>
<tr>
<td>x &lt;= y</td>
<td>x - y</td>
<td>x - y + 1</td>
</tr>
</tbody>
</table>
def testMe(x, y):
    if x == 2 * y and y > 1:
        return True
    else:
        return False
def testMe(x, y):
    if x <= y:
        if x == y:
            print("Some output")
        if x > 0:
            if y == 17:
                # Target Branch
                return True
            return False
    return False
def testMe(x, y):
    if x <= y:
        if x == y:
            print("Some output")
        if x > 0:
            if y == 17:
                # Target Branch
                return True
    return False
Covering a structure
Fitness evaluation

The test data executes the ‘wrong’ path
Approach Level

TARGET = 2
TARGET = 1
TARGET = 0

minimisation
Putting it all together

Fitness = approach Level + normalised branch distance

```c
void f1(int a, int b, int c, int d) {
    if (a > b) {
        if (b > c) {
            if (c > d) {
                // target
            }
        } else {
            // target
        }
    }
    ...
}
```

normalised branch distance between 0 and 1 indicates how close approach level is to being penetrated
Evolutionary Testing

- Mutation
- Crossover
- Selection
- Insertion
- Fitness Evaluation
- Test cases
- Monitoring
- Execution
- End?
void test_me(int a, int b, int c, int d) {
    if (a == b) {
        if (c == d) {
            // branch we want to execute
        }
    }
    ...
}
Crossover

```c
void test_me(int a, int b, int c, int d) {
    if (a == b) {
        if (c == d) {
            // branch we want to execute
        }
    }
    ...
}
```
Mutation

```java
void test_me(int a, int b, int c, int d) {
    if (a == b) {
        if (c == d) {
            // branch we want to execute
        }
    }
    ...
}
```
Selection

• Selective pressure:
The higher, the more likely the fittest are chosen

• Stagnation:
Selective pressure too small

• Premature convergence:
Selective pressure too high

• Standard algorithms:
Rank selection, tournament selection, roulette wheel selection
Contents

1. What is Search Based Software Testing?
2. Building an SBST Tool is Easy!
3. The EvoSuite Test Generation Tool
4. Lessons Learned Building an SBST Tool
@Test

public void test()
{
    int x = 2;
    int y = 2;
    int result = x + y;
    assertEquals(4, result);
}
@Test

public void test() {
    int var0 = 10
    YearMonthDay var1 = new YearMonthDay(var0);
    TimeOfDay var2 = new TimeOfDay();
    DateTime var3 = var1.toDateTime(var2);
    DateTime var4 = var3.minus(var0);
    DateTime var5 = var4.plusSeconds(var0);
}

Test Suite Generation

- Initialize Population
- While not done
  - Select parents
  - Recombine parents
- Return best solution

```java
int var0 = 10
YearMonthDay var1 = new YearMonthDay(var0);
TimeOfDay var2 = new TimeOfDay();
DateTime var3 = var1.toDateTime(var2);
DateTime var4 = var3.minus(var0);
TimeOfDay var2 = new TimeOfDay();
YearMonthDay var1 = new YearMonthDay(var0);
DateTime var5 = var4.plusSeconds(var0);
```
Test Suite Generation
Crossover
Mutation
public int gcd(int x, int y) {
    int tmp;
    while (y != 0) {
        tmp = x % y;
        x = y;
        y = tmp;
    }
    return x;
}
Components of an SBST Tool

- **Search Algorithm**: Genetic Algorithm (+Archive, Seeding, Local Search, DSE)
- **Representation**: Sets of sequences of Java statements
- **Search Operators**: Standard GA operators implemented for test suites
- **Fitness Function**: Sum of branch distances (and others)
- **Test Execution**: Java reflection
- **Instrumentation**: Java bytecode instrumentation
EvoSuite

http://www.evosuite.org/downloads

- Jar release - for command line usage
- Maven plugin
- IntelliJ plugin
- Eclipse plugin
- Jenkins plugin
Does it work?

SF110: 23,886 Classes  
6,628,619 LOC

Defects4J: 357 real bugs

Point is: It takes a tool and lots of engineering to do this.

G. Fraser, A. Arcuri. “A Large Scale Evaluation of Automated Unit Test Generation with EvoSuite” TOSEM 24(2), 2014.

Coverage

![Coverage Chart]

- **EvoSuite**
- **Manual**

- Option
- Rational
- DocType
- ArrayIntList
Time Spent on Testing

- Assisted
- Manual

- FilterIterator
- FixedOrderComparator
- ListPopulation
- PredicatedMap
Fault Detection

- Option
- Rational
- DocType
- ArrayIntList

EvoSuite vs Manual
Faults Prevention

- FilterIterator
- FixedOrderComparator
- ListPopulation
- PredicatedMap
@Test(timeout = 4000)
public void test3() throws Throwable {
    StringExample stringExample0 = new StringExample();
    boolean boolean0 = stringExample0.foo("");
    assertFalse(boolean0);
}

@Test(timeout = 4000)
public void testFooReturningFalse() throws Throwable {
    StringExample stringExample0 = new StringExample();
    boolean boolean0 = stringExample0.foo("");
    assertFalse(boolean0);
}
Variable Names

```java
@Test(timeout = 4000)
public void testFooReturningFalse() throws Throwable {
    StringExample stringExample0 = new StringExample();
    boolean boolean0 = stringExample0.foo("'");
    assertFalse(boolean0);
}
```

```java
@Test(timeout = 4000)
public void testFooReturningFalse() throws Throwable {
    StringExample invokesFoo = new StringExample();
    boolean resultFromFoo = invokesFoo.foo("'");
    assertFalse(resultFromFoo);
}
```
public class Foo {
    public void foo() {
        StringExample sx = new StringExample();
        boolean bar = sx.foo(""),
    }
}

@Test(timeout = 4000)
public void testFooReturningFalse() throws Throwable {
    StringExample sx = new StringExample();
    boolean bar = sx.foo(""),
    assertFalse(bar);
}
public void test3() throws Throwable {
    LongAdder longAdder0 = new LongAdder();
    longAdder0.reset();
    assertEquals(0, longAdder0.shortValue());
}
Time Spent Understanding

- StdXMLReader
- Attribute
- ChainBase
- Option
- FilterListIterator
- PluginRules
- RulesBase
- CharRange
- YearMonthDay

Time (min)

- Default
- Optimised
Contents

1. What is Search Based Software Testing?
2. Building an SBST Tool is Easy!
3. The EvoSuite Test Generation Tool
4. Lessons Learned Building an SBST Tool
1. Java

...is a weird language and never ceases to surprise me

My personal enemy: Java Generics

Bytecode over sourcecode - yes!
2. Corner Cases

The more corner cases you cover...
...the more can go wrong...
...the more new corner cases you will find...
...the slower EvoSuite becomes...
2. Corner Cases

- Constant Seeding: +5%
- Virtual FS: +1.4%
- Mocking: +4.7%
- JEE support: +3%
- DSE: +1.2%
3. Developers

...some really care only about coverage

...others don’t care about coverage:
"I wouldn’t normally in real life be aiming for 100% coverage. I’d probably end up with fewer tests without this tool but I couldn’t tell you if they would be all the right tests."

...do not want their tests to be generated

...hate ugly tests

...don’t like waiting

Talk to them!
3. Developers

```java
public class Example {

    private Example() {}

    // ...

}
```
4. Testing

Testing randomised algorithms is difficult

Make the implementation deterministic

Always use LinkedHashSet over HashSet,
LinkedHashMap over HashMap

Java reflection is not deterministic

Avoid static state (e.g. singletons)
4. Testing

EvoSuite uses one central random number generator

Any change will affect something at a completely different part of the program

Change seeds frequently during testing to find flaky tests
5. Documentation

I don’t comment my code

Students struggle

I spend more time explaining things than it would take me to implement them
6. Tool Comparisons

Reviewers want to see them
I don’t like doing them
It’s impossible to make them fair
Contact tool authors
Report bugs
Make your own tools usable
7. Open Source

“The source code will be released under an open source library (most likely GPL2) at a later point, as soon as a number of refactorings are completed.” — FSE’11 tool paper appendix

Public GitHub repo: 2015

It will never be clean enough, just release it!
8. Licensing

License matters

Google will not touch GPL

BSD, MIT - do you want others to become rich with your idea?

Gnu Lesser Public License, Apache
9. Tool Papers

The first one will be cited
The rest no one will cite
It shouldn’t be this way
10. Tool Building

Building a quick prototype is easy

…and will give you a paper

Building a real tool is difficult

…but lets you identify many new problems
…the lets you talk to developers
…the lets other people build on your work
…the will give you lots of citations and papers
10. Tool Building

Building a quick prototype is easy... and will give you a paper... but lets you identify many new problems... lets you talk to developers... lets other people build on your work... will give you lots of citations and papers.

Search-Based Software Engineers Need Tools!

...will give you lots of citations and papers.