“Searching” for the Best Tests

An Introduction to Automated Software Testing with Search-Based Techniques

Gregory M. Kapfhammer

Department of Computer Science
Allegheny College

March 31, 2015
Software is Everywhere

Software is pervasive — and so it must be thoroughly tested!
Software is Everywhere

Software is pervasive — and so it must be thoroughly tested!
Software is Everywhere
Software is pervasive — and so it must be thoroughly tested!

Program

Desktop Computer

Computer Server

Program

Mobile Computer
Software is Everywhere

Software is pervasive — and so it must be thoroughly tested!

- Program
  - Desktop Computer
  - Computer Server
  - Mobile Computer
  - Household Appliance
Software is Everywhere

Software is pervasive — and so it must be thoroughly tested!

<table>
<thead>
<tr>
<th>Program</th>
<th>Program</th>
<th>Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Desktop Computer</td>
<td>Computer Server</td>
<td>Mobile Computer</td>
</tr>
<tr>
<td>Scientific Device</td>
<td>Household Appliance</td>
<td></td>
</tr>
</tbody>
</table>
Software is Everywhere

Software is pervasive — and so it must be thoroughly tested!
Software is Complex

Even simple programs are intricate — and difficult to test!
Software is Complex

Even simple programs are intricate — and difficult to test!
Software is Complex

Even simple programs are intricate — and difficult to test!
Software is Complex

Even simple programs are intricate — and difficult to test!

Lines of Code

Numerous Features

Feature Interactions
Software is Complex
Even simple programs are intricate — and difficult to test!

[Diagram:
- Computer Software
  - Lines of Code
  - Numerous Features
  - Feature Interactions
  - Runtime Environns]
Software is Complex
Even simple programs are intricate — and difficult to test!

“Software entities are more complex for their size than perhaps any other human construct” — Frederick P. Brooks, Jr.

Diagram:
- Computer Software
  - Lines of Code
  - Numerous Features
  - Feature Interactions
  - Runtime Environos
Software is Evolving

Software is continuously updated — making testing critical!
Software is Evolving
Software is continuously updated — making testing critical!

Program
Execution Environment

Program
Execution Environment
Software is Evolving
Software is continuously updated — making testing critical!

Program Changed because of the addition of a new feature or the correction of a defect
Software is Evolving

Software is continuously updated — making testing critical!
Software is Evolving

Software is continuously updated — making testing critical!
Software is Evolving
Software is continuously updated — making testing critical!

Execution Environment Changed due to an upgrade in a kernel, device driver, or virtual machine
Software is Evolving

Software is continuously updated — making testing critical!

Execution Environment Changed due to an upgrade in a kernel, device driver, or virtual machine

“Release early, release often” means that programs are regularly updated
Motivating Example

The computation of an object’s velocity presents challenges!

\[ K = \frac{1}{2} m \times v^2 \]
Motivating Example

The computation of an object’s velocity presents challenges!

\[ K = \frac{1}{2} m \times v^2 \]
Motivating Example

The computation of an object’s velocity presents challenges!

\[ K = \frac{1}{2} m v^2 \]
Motivating Example

The computation of an object’s velocity presents challenges!

\[ K = \frac{1}{2} m \times v^2 \]
public static String computeVelocity(int kinetic, int mass) {
    int velocitySquared = 0;
    int velocity = 0;
    StringBuffer finalVelocity = new StringBuffer();
    if (mass != 0) {
        velocitySquared = 3 * (kinetic / mass);
        velocity = (int)Math.sqrt(velocitySquared);
        finalVelocity.append(velocity);
    } else {
        finalVelocity.append("Undefined");
    }
    return finalVelocity.toString();
}
Important Questions
Finding software defects is a challenging and rewarding task.

Can you find the defect in this program?
Important Questions
Finding software defects is a challenging and rewarding task

Are there general purpose strategies for defect isolation?
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development

Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
What is a Test Case?
A test case calls a method and checks its output with an oracle

Method Under Test
What is a Test Case?
A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.
What is a Test Case?

A test case calls a method and checks its output with an oracle.

The test case passes and the code is correct!
What is a Test Case?

A test case calls a method and checks its output with an oracle.

Diagram:
- **Input**
  - Method Under Test
    - Test Set Up
    - Output
    - Test Clean Up
  - Test Oracle
    - Expected Output
    - Test Verdict
What is a Test Case?

A test case calls a method and checks its output with an oracle.

The test case fails and a defect is found!
What is a Test Suite?

A test suite is an organized collection of test cases.
What is a Test Suite?
A test suite is an organized collection of test cases
What is a Test Suite?

A test suite is an organized collection of test cases
What is a Test Suite?

A test suite is an organized collection of test cases.
What is a Test Suite?

A test suite is an organized collection of test cases.
What is a Test Suite?
A test suite is an organized collection of test cases
What is a Test Suite?
A test suite is an organized collection of test cases.

Organize the Test Cases into a Test Suite

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow \ldots \rightarrow T_n$
What is a Test Suite?

A test suite is an organized collection of test cases

Organize the Test Cases into a Test Suite

Tool Support for Software Testing?
What is a Test Suite?

A test suite is an organized collection of test cases

Organize the Test Cases into a Test Suite

Tool Support for Software Testing?

JUnit
What is a Test Suite?

A test suite is an organized collection of test cases

Organize the Test Cases into a Test Suite

Tool Support for Software Testing?

JUnit  Apache Ant
What is a Test Suite?

A test suite is an organized collection of test cases.

Organize the Test Cases into a Test Suite

Tool Support for Software Testing?

JUnit  Apache Ant  Eclipse
A JUnit Test Case

```java
@Test
public void testOne() {
    String expected = new String("Undefined");
    String actual = Kinetic.
                       computeVelocity(5,0);
    assertEquals(expected, actual);
}
```
Another JUnit Test

```java
@Test
public void testTwo() {
    String expected = new String("0");
    String actual = Kinetic.computeVelocity(0, 5);
    assertEquals(expected, actual);
}
```
Important Questions
Not all tests have the same fault detection effectiveness!

Will these test cases find the fault in the example program?
Important Questions

Not all tests have the same fault detection effectiveness!

$T_1$ assigns

$K = 5, m = 0$ — Pass or fail?
Important Questions

Not all tests have the same fault detection effectiveness!

$T_2$ assigns

$K = 0, m = 5$ — Pass or fail?
Important Questions

Not all tests have the same fault detection effectiveness!

How do we study the effectiveness of different test cases?
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program’s data state
- Propagate to the program’s output
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program’s data state
- Propagate to the program’s output
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program’s data state
- Propagate to the program’s output
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program’s data state
- Propagate to the program’s output

All of these must occur before the fault manifests itself as a failure!
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program's data state
- Propagate to the program's output

All of these must occur before the fault manifests itself as a failure!
The PIE Model

There are necessary and sufficient conditions for fault detection

- Execute the faulty source code
- Infect the program’s data state
- Propagate to the program’s output

All of these *must occur* before the fault manifests itself as a failure!

Using the PIE model, will the test cases *find* the defect in the program?
A JUnit Test Case — $T_1$

```java
@Test
public void testOne() {
    String expected = new String("Undefined");
    String actual = Kinetic.computeVelocity(5, 0);
    assertEquals(expected, actual);
}
```
@Test
public void testOne() {
    String expected = new String("Undefined");
    String actual = Kinetic.computeVelocity(5,0);
    assertEquals(expected, actual);
}

E I P
X  X  X  X
@Test
public void testTwo() {
    String expected = new String("0");
    String actual = Kinetic.computeVelocity(0,5);
    assertEquals(expected, actual);
}

E I P
A JUnit Test Case — \( T_2 \)

```java
@Test
public void testTwo() {
    String expected = new String("0");
    String actual = Kinetic.computeVelocity(0, 5);
    assertEquals(expected, actual);
}
```
A JUnit Test Case — $T_3$

```java
@Test
public void testThree() {
    String expected = new String("4");
    String actual = Kinetic.computeVelocity(8, 1);
    assertEquals(expected, actual);
}
```
@Test
def testThree()
    String expected = new String("4");
    String actual = Kinetic.
        computeVelocity(8,1);
    assertEquals(expected, actual);
}
@Test
def testFour()
    String expected = new String("20");
    String actual = Kinetic.
        computeVelocity(1000,5);
    assertEquals(expected, actual);
@Test
public void testFour() {
    String expected = new String("20");
    String actual = Kinetic.
        computeVelocity(1000,5);
    assertEquals(expected, actual);
}

A test case must create specific inputs in order to cause failure!

Test Case | Status
---|---
$T_1$ | Pass
$T_2$ | Pass
$T_3$ | Pass
$T_4$ | Fail
I shall not deny that the construction of these testing programs has been a major intellectual effort: to convince oneself that one has not overlooked “a relevant state” and to convince oneself that the testing programs generate them all is no simple matter.

The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
  Test Cases
  Test Suites
  Examples of Tests
  The PIE Model
  Test Case Effectiveness

Search-Based Software Testing
  Testing Methods
  Random Testing
  Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development
  Pervasiveness of Software
  Complexity of Software
  Evolving Nature of Software
  Motivating Example
  Important Questions

Benefits of Software Testing
  Test Cases
  Test Suites
  Examples of Tests
  The PIE Model
  Test Case Effectiveness

Search-Based Software Testing
  Testing Methods
  Random Testing
  Testing with EvoSuite

Conclusion
Manual Testing

While it has benefits, this industry standard may be limited!
Manual Testing

While it has benefits, this industry standard may be limited!
Manual Testing

While it has benefits, this industry standard may be limited!
Manual Testing

While it has benefits, this industry standard may be limited!

- Laborious
- Time Consuming
- Very Tedious
Manual Testing
While it has benefits, this industry standard may be limited!

- Laborious
- Time Consuming
- Difficult
- Very Tedious
Manual Testing
While it has benefits, this industry standard may be limited!

Can we develop and employ methods that will automatically generate high-quality test cases for real-world software?
Automated Testing

Automatically generating tests is amazing — but does it work?
Automated Testing

Automatically generating tests is amazing — but does it work?
Automated Testing

Automatically generating tests is amazing — but does it work?

Automated Testing

Laborious

Time Consuming
Automated Testing

Automatically generating tests is amazing — but does it work?

- Laborious
- Time Consuming
- Very Tedious
Automated Testing

Automatically generating tests is amazing — but does it work?

Automated Testing
- Laborious
- Time Consuming
- Difficult
- Very Tedious
Automated Testing

Automatically generating tests is amazing — but does it work?

Testing is less laborious and tedious because an algorithm generates the tests. While computational time is needed, a human can be less involved!
Automated testing is less difficult since a good fitness function can guide the algorithm to inputs that find the faults.
Random Testing

It is easy to randomly generate tests — but how good are they?
Search-Based Testing

Use a fitness function to guide the search to “good” values
Mutation Testing

Let’s purposefully insert faults into the program under test!
Mutation Testing

Let’s purposefully insert faults into the program under test!
Mutation Testing

Let’s purposefully insert faults into the program under test!
Mutation Testing

Let’s purposefully insert faults into the program under test!
Mutation Testing

Let’s purposefully insert faults into the program under test!
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing
Let’s purposefully insert faults into the program under test!

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
Mutation Testing
Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{M_1, M_2, \ldots, M_{11}, M_{12} \}$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{ M_1, M_2, \ldots, M_{11}, M_{12} \}$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)

Set of Program Methods \( M = \{ M_1, M_2, \ldots, M_{11}, M_{12} \} \)
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{M_1, M_2, \ldots, M_{11}, M_{12}\}$
Mutation Testing
Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{M_1, M_2, \ldots, M_{11}, M_{12}\}$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{M_1, M_2, \ldots, M_{11}, M_{12} \}$
Mutation Testing
Let’s purposefully insert faults into the program under test!

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Set of Program Methods $M = \{M_1, M_2, \ldots, M_{11}, M_{12}\}$
Mutation Testing

Let’s purposefully insert faults into the program under test!

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)

Set of Program Methods \( M = \{ M_1, M_2, \ldots, M_{11}, M_{12} \} \)
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!

Evolutionary Testing

- Representation
- Fitness Function
- Modify Program
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!
Testing with EvoSuite

This prototype can automatically generate real JUnit test suites!

Configuring EvoSuite

This tool has many unique configurations — which are best?

EvoSuite’s Configurations
Configuring EvoSuite
This tool has many unique configurations — which are best?

EvoSuite’s Configurations

Random
Configuring EvoSuite
This tool has many unique configurations — which are best?

- Random
- Fixed-Random
Configuring EvoSuite
This tool has many unique configurations — which are best?

- Random
- Fixed-Random
- Genetic
Configuring EvoSuite

This tool has many unique configurations — which are best?

EvoSuite’s Configurations

- Random
- Fixed-Random
- Genetic
- Regression
Configuring EvoSuite

This tool has many unique configurations — which are best?

The fitness function computes branch coverage, weak mutation score, or strong mutation score to guide the search to the “best” test cases.
A Test from EvoSuite

@Test
class Test001 {
    public void test001() throws Throwable {
        int int0 = 0;
        String string0 = Kinetic.computeVelocity(int0, int0);
        assertEquals("Undefined", string0);
        assertNotNull(string0);
        int int1 = 1262;
        String string1 = Kinetic.computeVelocity(int0, int0);
        int int2 = 5349;
        String string2 = Kinetic.computeVelocity(int1, int2);
        int int3 = 0;
        int int4 = 3;
        String string3 = Kinetic.computeVelocity(int3, int4);
        Kinetic kinetic0 = new Kinetic();
    }
}
Important Questions

EvoSuite is an advanced, yet sometimes limited, testing tool

Will EvoSuite’s test cases find the fault in the example program?
Important Questions

EvoSuite is an advanced, yet sometimes limited, testing tool.

What is missing from the test cases that EvoSuite generates?
Important Questions

EvoSuite is an advanced, yet sometimes limited, testing tool.

How does the oracle problem influence the effectiveness of EvoSuite?
Important Questions

EvoSuite is an advanced, yet sometimes limited, testing tool.

What are the fundamental limitations of automated testing?
No Silver Bullet
Software tools are fundamentally limited — what is our hope?

There is no single development, in either technology or management technique, which by itself promises even one order-of-magnitude improvement within a decade in productivity, in reliability, in simplicity.

Frederick P. Brooks, Jr., *Proceedings of the IFIP Tenth World Computing Conference*, 1986
The Challenges of Software Development
  Pervasiveness of Software
  Complexity of Software
  Evolving Nature of Software
  Motivating Example
  Important Questions

Benefits of Software Testing
  Test Cases
  Test Suites
  Examples of Tests
  The PIE Model
  Test Case Effectiveness

Search-Based Software Testing
  Testing Methods
  Random Testing
  Testing with EvoSuite

Conclusion
The Challenges of Software Development
  Pervasiveness of Software
  Complexity of Software
  Evolving Nature of Software
  Motivating Example
  Important Questions

Benefits of Software Testing
  Test Cases
  Test Suites
  Examples of Tests
  The PIE Model
  Test Case Effectiveness

Search-Based Software Testing
  Testing Methods
  Random Testing
  Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Challenges of Software Development
Pervasiveness of Software
Complexity of Software
Evolving Nature of Software
Motivating Example
Important Questions

Benefits of Software Testing
Test Cases
Test Suites
Examples of Tests
The PIE Model
Test Case Effectiveness

Search-Based Software Testing
Testing Methods
Random Testing
Testing with EvoSuite

Conclusion
The Software Crisis
Solutions are available — but not always obvious or popular

What are the solutions to the software crisis?
The Software Crisis
Solutions are available — but not always obvious or popular

Great Designers
The Software Crisis
Solutions are available — but not always obvious or popular

Unremitting Care
The Software Crisis

Solutions are available — but not always obvious or popular

Incremental Advances
Final Admonishment

An epilogue called “Fifty Years of Wonder, Excitement, and Joy”

Too many interests, too many exciting opportunities for learning, research, and thought. What a marvelous predicament! Not only is the end not in sight, but the pace is not slackening. We have many future joys.

Frederick P. Brooks, Jr., The Mythical Man–Month, 1995