

SOFT620020.02

Advanced Software Engineering

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

<https://chenbihuan.github.io>

A Quick Recap

- Software testing
 - Unit testing, integration testing, system testing
- Black-box testing
 - Random testing, equivalence partitioning
- White-box testing
 - Coverage testing, mutation testing
- Grey-box testing

Discussion 1 – Testing OO Programs



Write test cases for the following two object-oriented programs.

```
java.util.HashSet<E>.add(E e)
```

```
public static void test() {  
    Set s = new HashSet();  
    s.add("hi");  
    assertTrue(s.equals(s));  
}
```

```
java.util.Date.setMonth(int month)
```

```
public static void test() {  
    Date d = new Date(2018, 9, 8);  
    d.setMonth(10);  
    assertTrue(d.equals(d));  
}
```

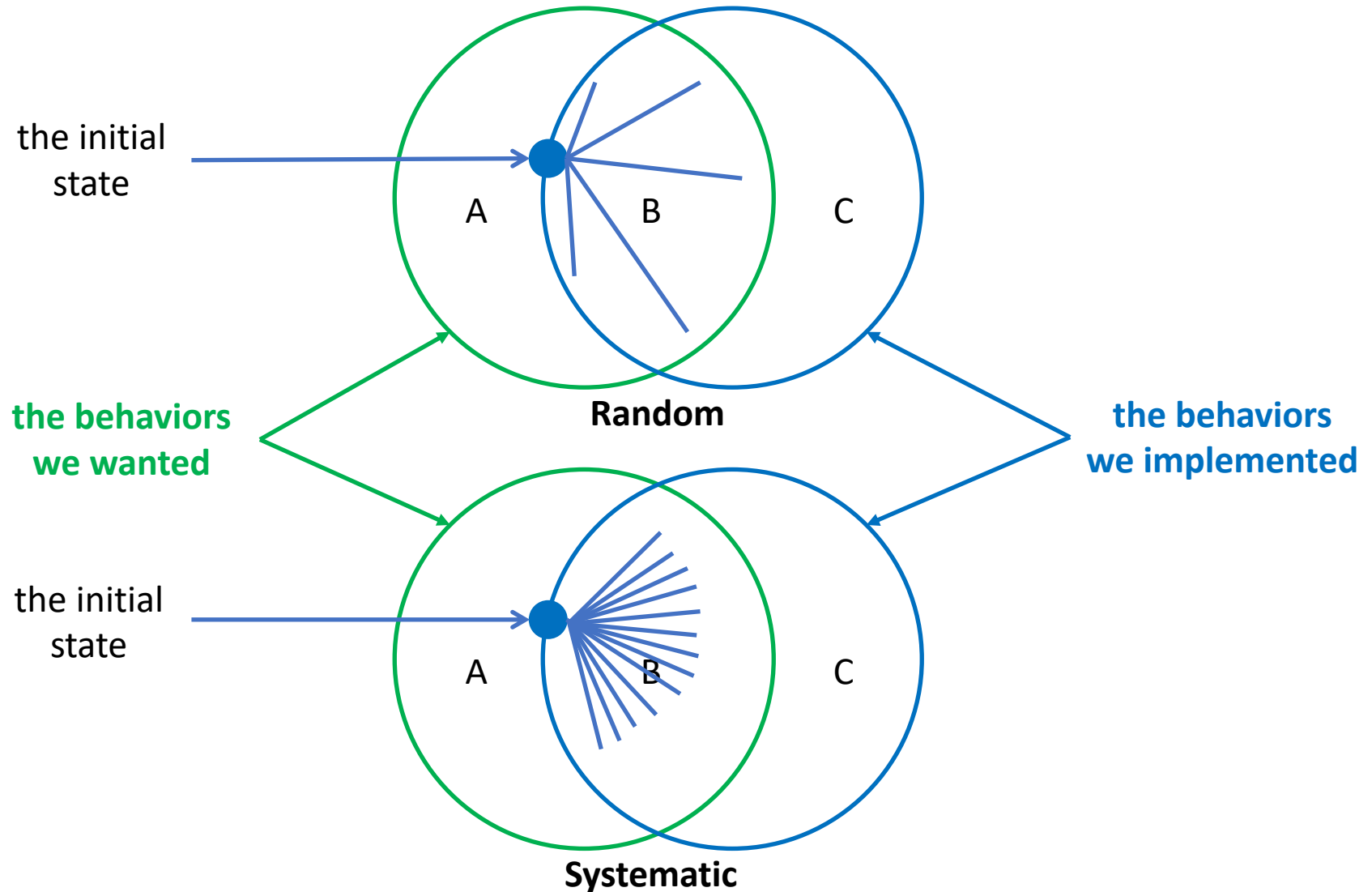
Course Outline

| Date | Topic | Date | Topic |
|---------|------------------------------|---------|--------------------|
| Sep. 10 | Introduction | Nov. 05 | Compiler Testing |
| Sep. 17 | Testing Overview | Nov. 12 | Mobile Testing |
| Sep. 24 | Holiday | Nov. 19 | Delta Debugging |
| Oct. 01 | Holiday | Nov. 26 | Presentation 1 |
| Oct. 08 | Guided Random Testing | Dec. 03 | Bug Localization |
| Oct. 15 | Search-Based Testing | Dec. 10 | Automatic Repair |
| Oct. 22 | Performance Analysis | Dec. 17 | Symbolic Execution |
| Oct. 29 | Security Testing | Dec. 24 | Presentation 2 |

Feedback-Directed Random Test Generation

Carlos Pacheco, Shuvendu Lahiri, Michael Ernst, Thomas Ball
ICSE 2007, Citation: 668

Random vs. Systematic



Random vs. Systematic (cont.)

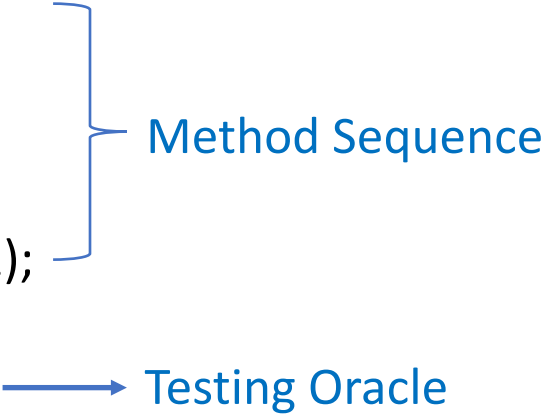
- Theoretical work suggests that random testing is as effective as more systematic testing techniques
 - Duran et al. 1984 and Hamlet et al. 1990
- Some empirical studies suggest that systematic testing is more effective than random testing
 - Ferguson et al. 1996: vs. chaining
 - Marinov et al. 2003: vs. bounded exhaustive
 - Visser et al. 2006: vs. model checking and symbolic execution
 - Studies are performed on small benchmarks, and do not measure error revealing effectiveness

RANDLOOP (**r**andom tester for **o**bject-**o**riented **p**rograms)

- Propose feedback-directed random test generation
 - Randomized creation of new test inputs is guided by feedback about the execution of previous test inputs
 - The goal is to generate **new** and **legal** test inputs and avoid **redundant** and **illegal** test inputs
- Conduct intensive empirical evaluations
 - Evaluate coverage **and error-detection capability** on a large number of widely-used and well-tested libraries (780K LOC)
 - Compare against systematic input generation
 - Compare against undirected random input generation

A Test Case Generated by RANDOOP

```
public static void test() {  
    LinkedList l1 = new LinkedList();  
    Object o1 = new Object();  
    l1.addFirst(o1);  
    TreeSet t1 = new TreeSet(l1);  
    Set s1 = Collections.unmodifiableSet(t1);  
    // this assertion fails  
    Assert.assertTrue(s1.equals(s1));  
}
```



Method Sequence

Testing Oracle

The set `s1` returned by `unmodifiableSet(Set)` returned false for `s1.equals(s1)`. This violates the reflexivity of `equals` as specified in Sun's API documentation.

The `TreeSet(Collection c)` constructor failed to throw `ClassCastException` as required by its specification (i.e., the elements in `c` must be `Comparable`)

Input and Output of RANDOOP

- Input
 - A set of classes under test
 - A set of (default or extended) contracts
 - A set of (default or extended) filters
 - Time limit (2 minutes by default)
- Output
 - A set of contract-satisfying test cases
 - A set of contract-violating test cases
- Main idea
 - Generate new test cases by extending previous ones
 - Execute the test case as soon as it is generated
 - Use the execution result to guide test case generation

Algorithm of RANDOOP

GenerateSequences(*classes, contracts, filters, timeLimit*)

```
1  errorSeqs  $\leftarrow \{\}$  // Their execution violates a contract.
2  nonErrorSeqs  $\leftarrow \{\}$  // Their execution violates no contract.
3  while timeLimit not reached do
4    // Create new sequence.
5     $m(T_1 \dots T_k) \leftarrow \text{randomPublicMethod}(\text{classes})$ 
6     $\langle \text{seqs}, \text{vals} \rangle \leftarrow \text{randomSeqsAndVals}(\text{nonErrorSeqs}, T_1 \dots T_k)$ 
7    newSeq  $\leftarrow \text{extend}(m, \text{seqs}, \text{vals})$ 
8    // Discard duplicates.
9    if newSeq  $\in \text{nonErrorSeqs} \cup \text{errorSeqs}$  then
10      continue
11    end if
12    // Execute new sequence and check contracts.
13     $\langle \vec{o}, \text{violated} \rangle \leftarrow \text{execute}(\text{newSeq}, \text{contracts})$ 
14    // Classify new sequence and outputs.
15    if violated = true then
16      errorSeqs  $\leftarrow \text{errorSeqs} \cup \{\text{newSeq}\}$ 
17    else
18      nonErrorSeqs  $\leftarrow \text{nonErrorSeqs} \cup \{\text{newSeq}\}$ 
19      setExtensibleFlags(newSeq, filters,  $\vec{o}$ ) // Apply filters.
20    end if
21  end while
22  return  $\langle \text{nonErrorSeqs}, \text{errorSeqs} \rangle$ 
```

Creating Method Sequence

- If T_i is a primitive type
 - Select a value from a fixed pool of values (e.g., -1, 0, 1, 'a', true)
 - The pool can be augmented by users
- If T_i is a reference type
 - Use a value v from a sequence that is already in *seqs*
 - Select a sequence from *nonErrorSeqs*, add it to *seqs*, and use a value v from it
 - Use *null*

Example of Creating Method Sequence

```
public class A {  
    public A() {...}  
    public B m1(A a1) {...}  
}
```

```
public class B {  
    public B(int i) {...}  
    public void m2(B b, A a) {...}  
}
```

One receiver object of type B, and two parameters of type B and A

| sequence s_1 | sequence s_2 | sequence s_3 |
|------------------|------------------|--------------------------------------|
| B b1 = new B(0); | B b2 = new B(0); | A a1 = new A(); B b3 = a1.m1(a1); |

| $seqs$ | $vals$ | $extend(m2, seqs, vals)$ |
|----------------------------|--|--|
| $\langle s_1, s_3 \rangle$ | $\langle s_1.1, s_1.1, s_3.1 \rangle$ (i.e.: b1, b1, a1) | B b1 = new B(0); A a1 = new A(); B b3 = a1.m1(a1); b1.m2(b1, a1); |
| $\langle s_1, s_2 \rangle$ | $\langle s_1.1, s_2.1, null \rangle$ (i.e.: b1, b2, null) | B b1 = new B(0); B b2 = new B(0); b1.m2(b2, null); |

Discussion 2 – Creating Sequences



Create ALL possible sequences for the method m2 in class B.

```
public class A {  
    public A() {...}  
    public B m1(A a1) {...}  
}
```

```
public class B {  
    public B(int i) {...}  
    public void m2(B b, A a) {...}  
}
```

Sequence s1

B b1 = new B(0);

Sequence s2

B b2 = new B(-1);

Sequence s3

A a1 = new A();

Algorithm of RANDOOP

GenerateSequences(*classes, contracts, filters, timeLimit*)

```
1  errorSeqs  $\leftarrow \{\}$  // Their execution violates a contract.
2  nonErrorSeqs  $\leftarrow \{\}$  // Their execution violates no contract.
3  while timeLimit not reached do
4    // Create new sequence.
5     $m(T_1 \dots T_k) \leftarrow \text{randomPublicMethod}(\text{classes})$ 
6     $\langle \text{seqs}, \text{vals} \rangle \leftarrow \text{randomSeqsAndVals}(\text{nonErrorSeqs}, T_1 \dots T_k)$ 
7    newSeq  $\leftarrow \text{extend}(m, \text{seqs}, \text{vals})$ 
8    // Discard duplicates.
9    if newSeq  $\in \text{nonErrorSeqs} \cup \text{errorSeqs}$  then
10      continue
11    end if
12    // Execute new sequence and check contracts.
13     $\langle \vec{o}, \text{violated} \rangle \leftarrow \text{execute}(\text{newSeq}, \text{contracts})$ 
14    // Classify new sequence and outputs.
15    if violated = true then
16      errorSeqs  $\leftarrow \text{errorSeqs} \cup \{\text{newSeq}\}$ 
17    else
18      nonErrorSeqs  $\leftarrow \text{nonErrorSeqs} \cup \{\text{newSeq}\}$ 
19      setExtensibleFlags(newSeq, filters,  $\vec{o}$ ) // Apply filters.
20    end if
21  end while
22  return  $\langle \text{nonErrorSeqs}, \text{errorSeqs} \rangle$ 
```

Discussion 3 – Redundant Sequence



Which method sequence is redundant?

```
Set t = new HashSet();  
s.add("hi");  
assertTrue(s.equals(s));
```

```
Set t = new HashSet();  
s.add("hi");  
assertTrue(s.equals(s));
```

```
Set t = new HashSet();  
s.add("hi");  
s.isEmpty();  
assertTrue(s.equals(s));
```


Redundancy Checking

- RANDOOP maintains a set of objects for each type
- A method sequence is redundant if the objects created during its execution are members of the above set
 - Use equals() to compare
 - Or user-defined more sophisticated checking
- RANDOOP uses **observer methods** to capture object state
 - A method is probably an observer method if it has no parameters; it is public and non-static; it returns primitive values; and its name is size, count, length, toString, or begins with get or is

Algorithm of RANDOOP

GenerateSequences(*classes, contracts, filters, timeLimit*)

```
1  errorSeqs  $\leftarrow \{\}$  // Their execution violates a contract.
2  nonErrorSeqs  $\leftarrow \{\}$  // Their execution violates no contract.
3  while timeLimit not reached do
4    // Create new sequence.
5     $m(T_1 \dots T_k) \leftarrow \text{randomPublicMethod}(\text{classes})$ 
6     $\langle \text{seqs}, \text{vals} \rangle \leftarrow \text{randomSeqsAndVals}(\text{nonErrorSeqs}, T_1 \dots T_k)$ 
7    newSeq  $\leftarrow \text{extend}(m, \text{seqs}, \text{vals})$ 
8    // Discard duplicates.
9    if newSeq  $\in \text{nonErrorSeqs} \cup \text{errorSeqs}$  then
10      continue
11    end if
12    // Execute new sequence and check contracts.
13     $\langle \vec{o}, \text{violated} \rangle \leftarrow \text{execute}(\text{newSeq}, \text{contracts})$ 
14    // Classify new sequence and outputs.
15    if violated = true then
16      errorSeqs  $\leftarrow \text{errorSeqs} \cup \{\text{newSeq}\}$ 
17    else
18      nonErrorSeqs  $\leftarrow \text{nonErrorSeqs} \cup \{\text{newSeq}\}$ 
19      setExtensibleFlags(newSeq, filters,  $\vec{o}$ ) // Apply filters.
20    end if
21  end while
22  return  $\langle \text{nonErrorSeqs}, \text{errorSeqs} \rangle$ 
```

Default Contracts in RANDOOP

| Method | |
|------------------|---|
| contract | description |
| Exception (Java) | method throws no <code>NullPointerException</code> if no input parameter was null |
| | method throws no <code>AssertionError</code> |
| Exception (.NET) | method throws no <code>NullReferenceException</code> if no input parameter was null |
| | method throws no <code>IndexOutOfRangeException</code> |
| | method throws no <code>AssertionError</code> |
| Object | |
| contract | description |
| equals | <code>o.equals(o)</code> returns true |
| | <code>o.equals(o)</code> throws no exception |
| hashCode | <code>o.hashCode()</code> throws no exception |
| toString | <code>o.toString()</code> throws no exception |

Default Filters in RANDOOP

- Filters determine which objects created in a method sequence can be reused as the input to a new method call
 - Equality
 - Null
 - Exception

The Equality Filter

- Use the equals() method to determine whether the resulting object has been created before
- Maintain a set of all reusable objects that have been created across all the sequences

The Null Filter

- Determine whether the resulting object is null
- Null dereference exception occur in the absence of any null value in the inputs indicates some internal problem with the method

The Exception Filter

- Exceptions frequently correspond to pre-condition violations for a method call, and therefore there is little point reusing them
- An extension of the sequence would lead to an exception before the execution completes

Discussion 4 – Filters



Which method sequence or object is filtered?

```
Date d = new Date(2006, 2, 14);  
d.setMonth(1);  
assertTrue(d.equals(d));
```

```
Date d = new Date(2006, 2, 14);  
// pre-condition: argument >= 0  
d.setMonth(-1);  
assertTrue(d.equals(d));
```

```
Date d = new Date(2006, 2, 14);  
d.setMonth(-1);  
d.setDay(5);  
assertTrue(d.equals(d));
```

```
Object o = returnNull();  
LinkedList l = new LinkedList();  
l.add(o);
```


Repetition

- Repeated calls may be necessary
 - Reach the code that increases the capacity of a container object
 - Create two equivalent objects that can cause a method like equals to go down certain branches
- When generating a new sequence, with probability p , RANDOOP appends m calls of a chosen method
 - p is in the range of 0 and 1, its default value is 0.1
 - m is chosen uniformly at random between 0 and max , and the default value of max is 100

Discussion 5 – Feedback in RANDOOP



What are the feedbacks used in RANDOOP?

Contracts and Filters

Evaluation – Testing Containers

| | | coverage | | | | time (seconds) | | | |
|-------------------------------------|---------|----------|-----|------------------|-----------------|----------------|------|------------------|-----------------|
| | | JPF | RP | JPF _U | RP _U | JPF | RP | JPF _U | RP _U |
| <i>block</i> <i>coverage</i> | BinTree | .78 | .78 | .78 | .78 | 0.14 | 0.21 | 0.14 | 0.13 |
| | BHeap | .95 | .95 | .95 | .86 | 4.3 | 0.59 | 6.2 | 6.6 |
| | FibHeap | 1 | 1 | 1 | .98 | 23 | 0.63 | 1.1 | 27 |
| | TreeMap | .72 | .72 | .72 | .68 | 0.65 | 0.84 | 1.5 | 1.9 |
| <i>predicate</i> <i>coverage</i> | BinTree | 53.2 | 54 | 52.1 | 53.9 | 0.41 | 1.6 | 2.0 | 4.2 |
| | BHeap | 101 | 101 | 88.3 | 58.5 | 9.8 | 4.2 | 12 | 15 |
| | FibHeap | 93 | 96 | 86 | 90.3 | 95 | 6.0 | 16 | 67 |
| | TreeMap | 106 | 106 | 104 | 55 | 47 | 10 | 10 | 1.9 |

JPF : Best-performing of 5 systematic techniques in JPF.

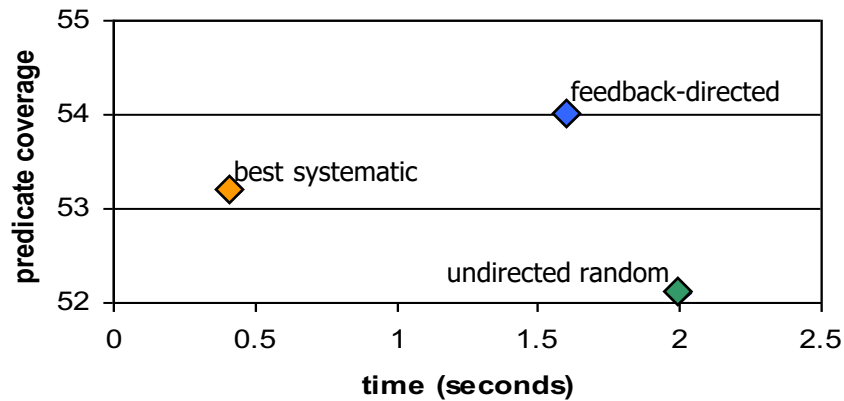
RP : RANDOOP: Feedback-directed random testing.

JPF_U : Undirected random testing implemented in JPF.

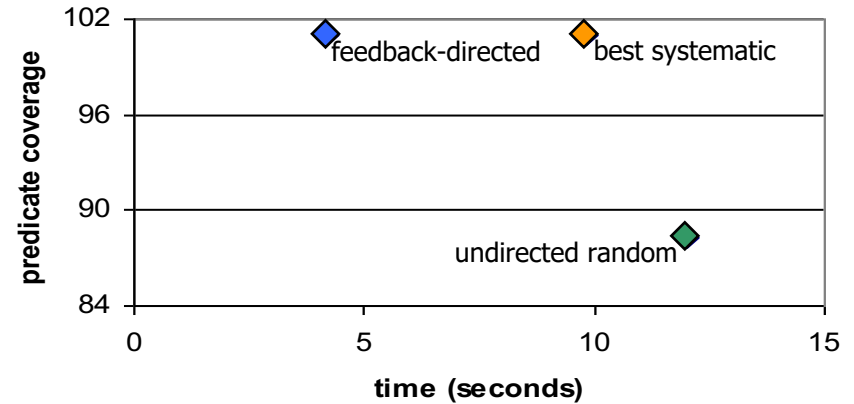
RP_U : Undirected random testing implemented in RANDOOP.

Evaluation – Testing Containers (cont.)

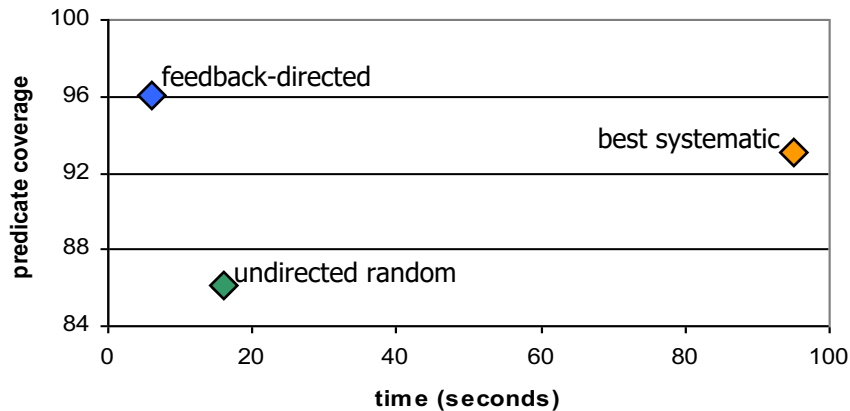
Binary tree



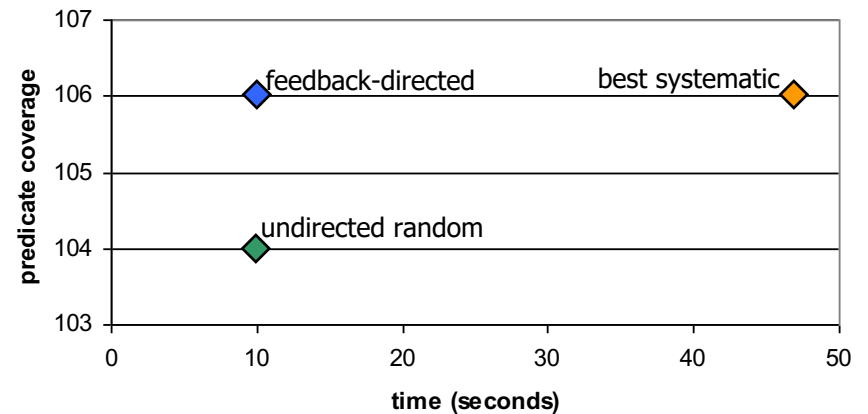
Binomial heap



Fibonacci heap



Tree map



Evaluation – Checking API Contracts

| Java libraries | LOC | public classes | public methods | description |
|-----------------|------|----------------|----------------|---|
| Java JDK 1.5 | | | | |
| java.util | 39K | 204 | 1019 | Collections, text, formatting, etc. |
| javax.xml | 14K | 68 | 437 | XML processing. |
| Jakarta Commons | | | | |
| chain | 8K | 59 | 226 | API for process flows. |
| collections | 61K | 402 | 2412 | Extensions to the JDK collections. |
| jelly | 14K | 99 | 724 | XML scripting and processing. |
| logging | 4K | 9 | 140 | Event-logging facility. |
| math | 21K | 111 | 910 | Mathematics and statistics. |
| primitives | 6K | 294 | 1908 | Type-safe collections of primitives. |
| | | | | |
| .NET libraries | LOC | public classes | public methods | |
| ZedGraph | 33K | 125 | 3096 | Creates plots and charts. |
| .NET Framework | | | | |
| Mscorlib | 185K | 1439 | 17763 | .NET Framework SDK class libraries. Provide access to system functionality and designed as foundation on which .NET applications, components, and controls are built. |
| System.Data | 196K | 648 | 11529 | |
| System.Security | 9K | 128 | 1175 | |
| System.Xml | 150K | 686 | 9914 | |
| Web.Services | 42K | 304 | 2527 | |

14 widely-used libraries comprising a total of 780K LOC

Evaluation – Checking API Contracts (cont.)

| library | test cases generated | violation-inducing test cases | REDUCE reported test cases | error-revealing test cases | errors | errors per KLOC |
|-----------------|----------------------|-------------------------------|----------------------------|----------------------------|--------|-----------------|
| Java JDK | | | | | | |
| java.util | 22,474 | 298 | 20 | 19 | 6 | .15 |
| javax.xml | 15,311 | 315 | 12 | 10 | 2 | .14 |
| Jakarta Commons | | | | | | |
| chain | 35,766 | 1226 | 20 | 0 | 0 | 0 |
| collections | 16,740 | 188 | 67 | 25 | 4 | .07 |
| jelly | 18,846 | 1484 | 78 | 0 | 0 | 0 |
| logging | 764 | 0 | 0 | 0 | 0 | 0 |
| math | 3,049 | 27 | 9 | 4 | 2 | .09 |
| primitives | 49,789 | 119 | 13 | 0 | 0 | 0 |
| ZedGraph | 8,175 | 15 | 13 | 4 | 4 | .12 |
| .NET Framework | | | | | | |
| mscorlib | 5,685 | 51 | 19 | 19 | 19 | .10 |
| System.Data | 8,026 | 177 | 92 | 92 | 92 | .47 |
| System.Security | 3,793 | 135 | 25 | 25 | 25 | 2.7 |
| System.Xml | 12,144 | 19 | 15 | 15 | 15 | .10 |
| Web.Services | 7,941 | 146 | 41 | 41 | 41 | .98 |
| Total | 208,503 | 4200 | 424 | 254 | 210 | |

JPF ran out of memory without reporting any errors for all the libraries.

Undirected Random found 60 errors, and did not find errors in java.util or javax.xml

Errors Found: Examples

- JDK Collections classes have 4 methods that create objects violating `o.equals(o)` contract
- Javax.xml creates objects that cause `hashCode` and `toString` to crash, even though objects are well-formed XML constructs
- Apache libraries have constructors that leave fields unset, leading to NPE on calls of `equals`, `hashCode` and `toString` (this only counts as one bug)
- Many Apache classes require a call of an *init()* method before object is legal—led to many false positives
- .Net framework has at least 175 methods that throw an exception forbidden by the library specification (NPE, out-of-bounds, of illegal state exception)
- .Net framework has 8 methods that violate `o.equals(o)`
- .Net framework loops forever on a legal but unexpected input

Evaluation – Regression testing

- Generated test cases using JDK 1.5
 - RANDOOP generated 41K regression test cases
- Ran resulting test cases on
 - JDK 1.6 Beta
 - 25 test cases failed
 - Sun's implementation of the JDK
 - 73 test cases failed
 - Failing test cases pointed to 12 distinct errors
 - These errors were not found by the extensive compliance test suite that Sun provides to JDK developers

Evaluation Summary

- Feedback-directed random test generation
 - Is effective at finding errors
 - Discovered several errors in real code (e.g. JDK, .NET framework core libraries)
 - Can outperform systematic input generation
 - On previous benchmarks and metrics (**coverage**)
 - On a new, larger corpus of subjects, measuring **error detection**
 - Can outperform undirected random test generation

Conclusion

- Feedback-directed random test generation
 - Finds errors in widely-used, well-tested libraries
 - Can outperform systematic test generation
 - Can outperform undirected test generation
- RANDOOP
 - Easy to use — just point at a set of classes
 - Has real clients: used by product groups at Microsoft
- A mid-point in the systematic-random space of input generation techniques

Discussion 6 – Improving RANDOOP



How do we improve RANDOOP?

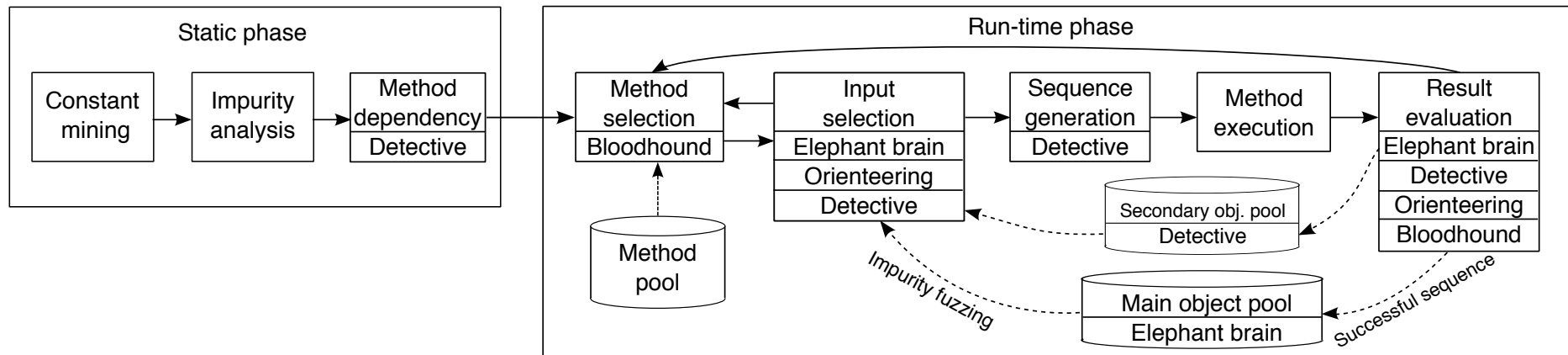
GRT: Program-Analysis-Guided Random Testing

Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Johannes
Gmeiner, Rudolf Ramlar

ASE 2015, ACM SIFSOFT Distinguish Paper Award

Overview of GRT

| Component | Static / dynamic | Description |
|-----------------|------------------|---|
| Constant mining | static | Extract constants from SUT for both global usage (seed the main object pool) and for local usage as inputs for specific methods. |
| Impurity | static + dynamic | Perform static purity analysis for all MUTs. At run-time, fuzz selected input from the object pool based on a Gaussian distribution and purity results. |
| Elephant brain | dynamic | Manage method sequences (to create inputs) in the object pool with the exact types obtained at run-time. |
| Detective | static + dynamic | Analyze the method input and return type dependency statically, and construct missing input data on demand at run-time. |
| Orienteering | dynamic | Favor method sequences that require lower cost to execute, which accelerates testing for other components. |
| Bloodhound | dynamic | Guide method selection and sequence generation by coverage information at run-time. |



Constant Mining

```
package org.apache.commons.cli;
public class PatternOptionBuilder{
    public static final Class STRING_VAL=String.class;
    public static final Class OBJECT_VAL=Object.class;
    public static final Class NUMBER_VALUE = Number.class;
    // 6 more similar fields omitted.
    public static Object getValueClass(char ch) {
        switch (ch) {
            case '@':return PatternOptionBuilder.OBJECT_VAL;
            case ':':return PatternOptionBuilder.STRING_VAL;
            case '%':return PatternOptionBuilder.NUMBER_VALUE;
            // 6 more case branches omitted.
        }
        return null;
    } // 2 more methods omitted.
}
public class TypeHandler {
    // 1 method omitted.
    public static Object createVal(String s, Class c) {
        if (PatternOptionBuilder.STRING_VAL == c)
            return s;
        else if (PatternOptionBuilder.OBJECT_VAL == c)
            return createObject(s);
        else if (PatternOptionBuilder.NUMBER_VALUE == c)
            return createNumber(s);
        // 6 more else if branches omitted.
        else return null; } // 7 more methods omitted.
```

Impurity Analysis: Object State Fuzzing

- Primitive value fuzzing
 - When a primitive number c is selected as an input, we adopt a Gaussian distribution to probabilistically fuzz its value and use the altered result as the input
- Reference value fuzzing
 - Gather all impure methods that can change the state of a reference object o
 - Randomly select an impure method m , and invoke m on o

Demand-Driven Input Construction

```
public class A {  
    public A() {...}  
    public B m1(A a1) {...}  
}  
  
public class B {  
    public B(int i) {...}  
    public void m2(B b, A a) {...}  
}
```

- Search constructors and methods that return an object of the required type
 - e.g., `A()`, `m1(A)`, `B(int)`
- Choose a method *m*, and recursively search for inputs needed to execute *m*
 - e.g., `B(int)`, `A()`
- Combine *m* with the method under test
 - e.g., `B b = new B(0); A a = new A(); b.m2(b, a);`

Cost-Guided Sequence Selection

- Multiple sequences can return the object of the same type
- Randomly selecting a method sequence as the input makes the generated method sequence grow considerably in length and execution cost
- Select a method sequence based on its execution cost

Coverage-Guided Method Selection

- An equally balanced selection of method user test wastes time on methods that are already well covered.
- Too much emphasis on method user test containing uncovered branches may waste time in challenging the difficult branches
- Design a weight function based on both code coverage and the execution history of each method under test

Reading Materials

- Carlos Pacheco, Shuvendu K. Lahiri, Michael D. Ernst, and Thomas Ball. 2007. Feedback-Directed Random Test Generation. In ICSE. 75–84.
- Carlos Pacheco, Shuvendu K. Lahiri, and Thomas Ball. 2008. Finding Errors in .Net with Feedback-directed Random Testing. In ISSTA. 87–96.
- Lei Ma, Cyrille Artho, Cheng Zhang, Hiroyuki Sato, Johannes Gmeiner, and Rudolf Ramlér. 2015. GRT: Program- Analysis-Guided Random Testing. In ASE. 212–223.
- Kohsuke Yato, Kazunori Sakamoto, Fuyuki Ishikawa, and Shinichi Honiden. 2015. Feedback-controlled Random Test Generation. In ISSTA. 316–326.
- Suresh Thummalapenta, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, and Wolfram Schulte. 2009. MSeqGen: Object-oriented Unit-test Generation via Mining Source Code. In ESEC/FSE. 193–202.
- Suresh Thummalapenta, Tao Xie, Nikolai Tillmann, Jonathan de Halleux, and Zhendong Su. 2011. Synthesizing Method Sequences for High-coverage Testing. In OOPSLA. 189–206.
- Sai Zhang, David Saff, Yingyi Bu, and Michael D. Ernst. 2011. Combined Static and Dynamic Automated Test Generation. In ISSTA. 353–363.
- Wujie Zheng, Qirun Zhang, Michael Lyu, and Tao Xie. 2010. Random Unit-test Generation with MUT-aware Sequence Recommendation. In ASE. 293–296.

Q&A?

Bihuan Chen, Pre-Tenure Assoc. Prof.

bhchen@fudan.edu.cn

<https://chenbihuan.github.io>