# Feedback-directed Random Test Generation

Pacheco, Lahiri, Ernst, Ball.

Presentation by Eric Vaughn

# Conventional Notions About Effectiveness of Random Testing

- Not as efficient as systematic techniques: chaining, exhaustive generation, model checking,  and symbolic execution.
- Achieves less code coverage.
- Much slower.
- Blackbox vs. Whitebox and Guided vs. Unguided Random Testing

# What the Randoop Tool does

- “Feedback-directed” Random unit test generator for Java (also used by Microsoft for testing .NET framework).
- Only one input for the tool - a class listing.
- Builds a logical and legal sequence of method calls/object constructors and asserts a condition about the final output at the end.

# Principles for generating a test sequence

- Classes and methods are randomly chosen and put together logically.
- If needed, methods and constructors are given constants or appropriate inputs from previous calls.
- These calls are chained together and executed.
- After each method call, the state of the object is checked (null ptr, reflexive equals, etc.).

# What does a test case look like?

- How is it “feedback-directed?”

```
public static void test1()
{
        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));
}
```

This was a test on a JDK class.

# Benefits of "feedback-directed" random testing

- Benefits over undirected random testing
- Avoids the generation of meaningless or redundant inputs.
- Benefits over systematic testing
- Not nearly as resource intensive:  Executes only small units of the program in a normal fashion.
- Each test, new classes and methods are chosen.
  - Rather than testing a localized area, we sample the whole program.

# Results of some case studies

| 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 | |

Figure 7. Statistics for test cases generated by RANDOOP. Section 3.2.1 explains the metrics.

# Conclusions

- Feedback-directed testing scales very well with large code.
- Its performance is on-par with and superior to several popular systematic testing techniques in both code coverage and efficiency.