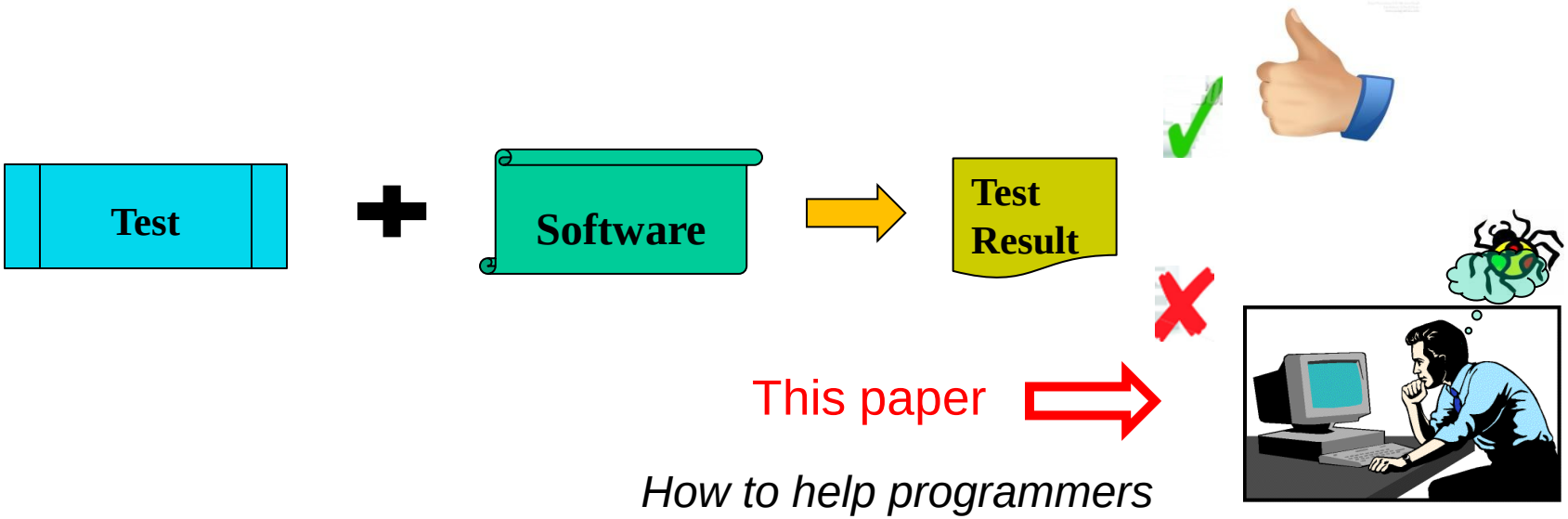
Practical Semantic Test Simplification

Sai Zhang
University of Washington

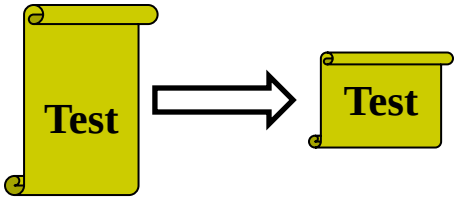
A typical testing workflow



How to help programmers understand a failed test?

Test simplification

- Isolate only the failure-relevant code
- State of the art:
  - Delta debugging [Zeller'02]
  - Slicing [Weiser'84]



- **Fundamental limitations**
  - Carve a subset of the existing test code
  - Simplify the test at the syntax level

An example failed test

(automatically generated by Randoop [Pacheco'07])

```
public void testTreeSet() {
  Object obj = new Object();
  Integer int1 = 0;
  Integer int2 = 1;
  Object[] objs = new Object[] {obj, int1, int2};
  List list1 = Arrays.asList(objs);
  List list2 = list1.sublist(int1, int2);
  TreeSet ts = new TreeSet();
  ts.add(list2);
  Set set = Collections.synchronizedSet(ts);
  assertTrue(set.equals(set));
}
```

- Syntactically-minimized
  - Both Delta debugging and slicing can no longer simplify it

The SimpleTest algorithm

- An algorithm simplifying tests at the semantic level
  - Preserve the testing oracle's behavior

- Key steps:
  - **Replace** a referred variable with other alternatives (**greedy**: use the earliest-defined type-compatible variable)
  - **Simplify** the test code by removing redundant code
  - **Validate** the oracle's behavior

```
Object a = null;
Object b = a;
Object c = b;
c.hashCode();
```

replacement ⇒

```
Object a = null;
Object b = a;
Object c = b;
a.hashCode();
```

simplification ⇒

```
Object a = null;
~~Object b = a;~~
~~Object c = b;~~
a.hashCode();
```

validation by execution ⇒

```
Object a = null;
a.hashCode();
```

Simplified!

Simplifying the example test

```
public void testTreeSet() {
  Object obj = new Object();
  Integer int1 = 0;
  Integer int2 = 1;
  Object[] objs = new Object[] {obj, int1, int2};
  List list1 = Arrays.asList(objs);
  List list2 = list1.sublist(int1, int2);
  TreeSet ts = new TreeSet();
  ts.add(list2);
  Set set = Collections.synchronizedSet(ts);
  assertTrue(set.equals(set));
}
```

Simplifying the example test

```
public void testTreeSet() {
  Object obj = new Object();
  Integer int1 = 0;
  Integer int2 = 1;
  Object[] objs = new Object[] {obj, int1, int2};
  List list1 = Arrays.asList(objs);
  List list2 = list1.sublist(int1, int2);
  TreeSet ts = new TreeSet();
  ts.add(obj);
  Set set = Collections.synchronizedSet(ts);
  assertTrue(set.equals(set));
}
```

Simplifying the example test

```
public void testTreeSet() {
  Object obj = new Object();
  Integer int1 = 0;
  Integer int2 = 1;
  Object[] objs = new Object[] {obj, int1, int2};
  List list1 = Arrays.asList(objs);
  List list2 = list1.sublist(int1, int2);
  TreeSet ts = new TreeSet();
  ts.add(obj);
  Set set = Collections.synchronizedSet(ts);
  assertTrue(set.equals(set));
}
```

A semantically-simplified test

```
public void testTreeSet() {
  Object obj = new Object();
  TreeSet ts = new TreeSet();
  ts.add(obj);
  Set set = Collections.synchronizedSet(ts);

  assertTrue(set.equals(set));
}
```

More in the paper

- The semantic test simplification problem
  - Formal definition
  - NP-Completeness proof

- SimpleTest algorithm
  - Complexity analysis
  - Optimizations to avoid trivial test like:

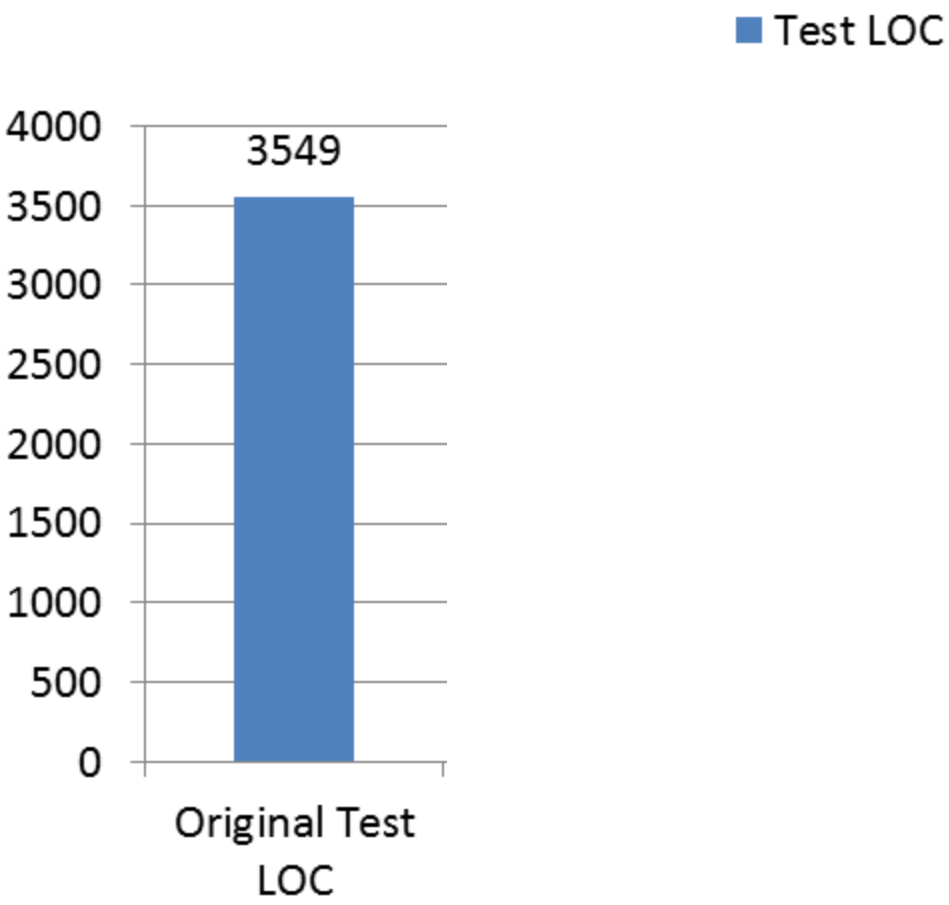
```
assert(null != null);
```

Preliminary experiments

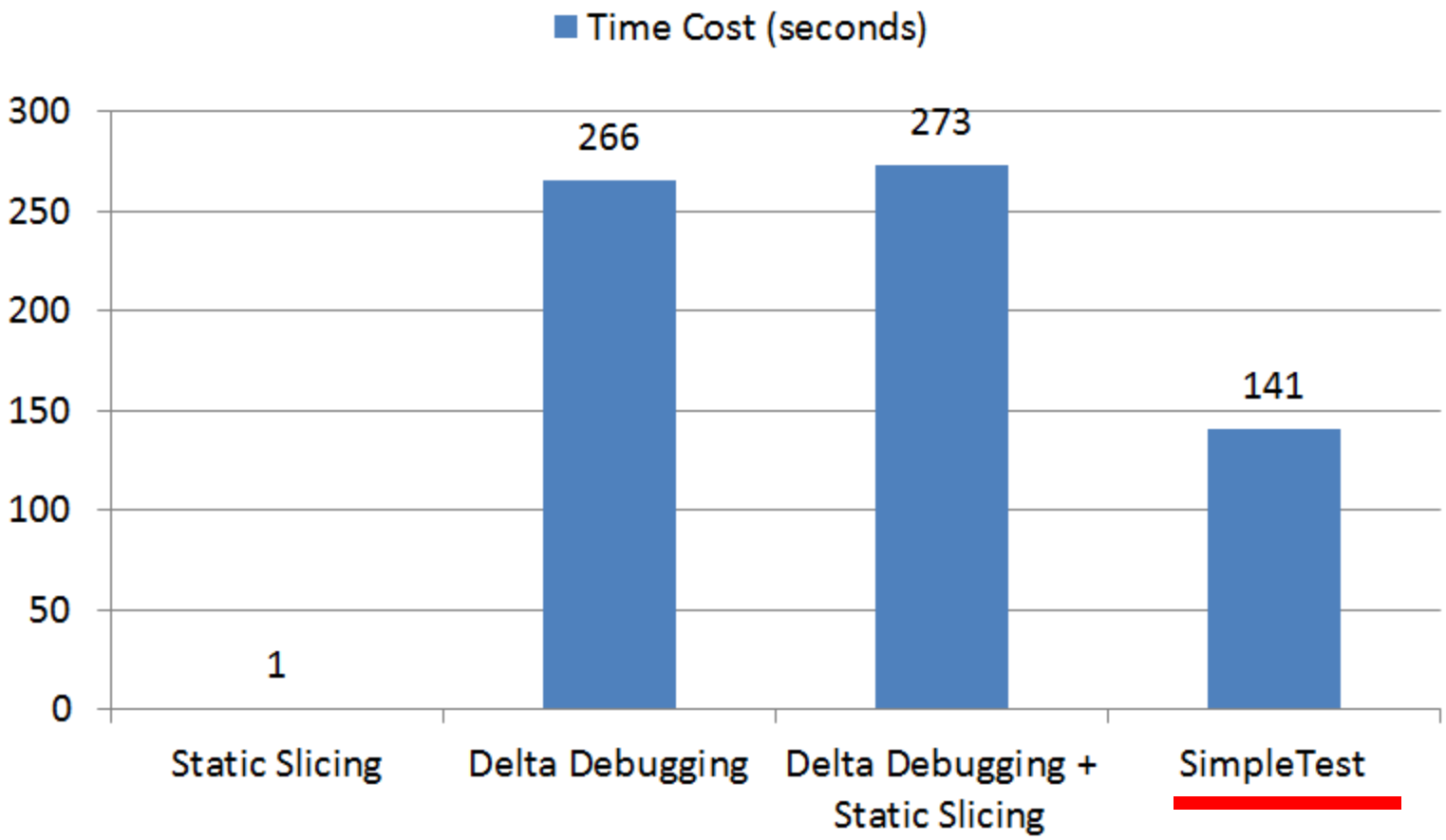
Subject Program	LOC	#Failed Tests	Failed Test LOC
Time And Money	2,372	23	1337
jdom	8,513	3	194
Apache Commons Primitives	9,368	5	377
Apache Commons Beanutils	11,382	10	317
Apache Commons Math	12,469	18	747
Apache Commons Collections	55,400	8	399
`java.util` package	48,026	6	178
Total	**149, 557**	**73**	**3549**

All tests are automatically generated by Randoop [Pacheco'07]

Results in simplifying failed tests



Time cost in simplifying failed tests



Related work

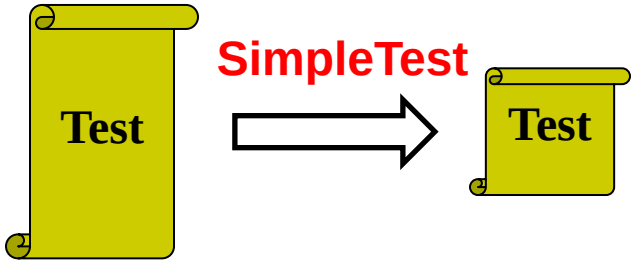
- Syntax-level test simplification
  - Delta debugging [Zeller'02]
  - Program slicing [Weiser'84]
  - Combined approach [Leitner'07]

  Will not be useful if a test is already syntactically-minimized

- Semantic-level **input value** reduction
  - Hierarchical delta debugging [Misherghi'06]
  - C-Reducer [Regehr'12]

  Does not preserve the semantics of the original test code

Contributions



- The semantic test simplification problem
  - Formal definition
  - Proof of its NP-Completeness
- A technique to semantically simplify tests
  - Uses a greedy algorithm
  - Fully automated
  - Preserves the behavior of a given oracle
- Experiments that demonstrate its usefulness
  - Outperforms existing syntax-level test simplification techniques

[Backup slides]

Future work

- Extending SimpleTest to handle more Java features
  - If-else conditions
  - Loops
  - Exceptions
- Comparing SimpleTest with other techniques
  - Dynamic slicing [Agrawal'91]
  - C-Reducer [Regehr'12]
- Exploring downstream applications of SimpleTest
  - Fault localization with the simplified tests
  - Explanatory document inference