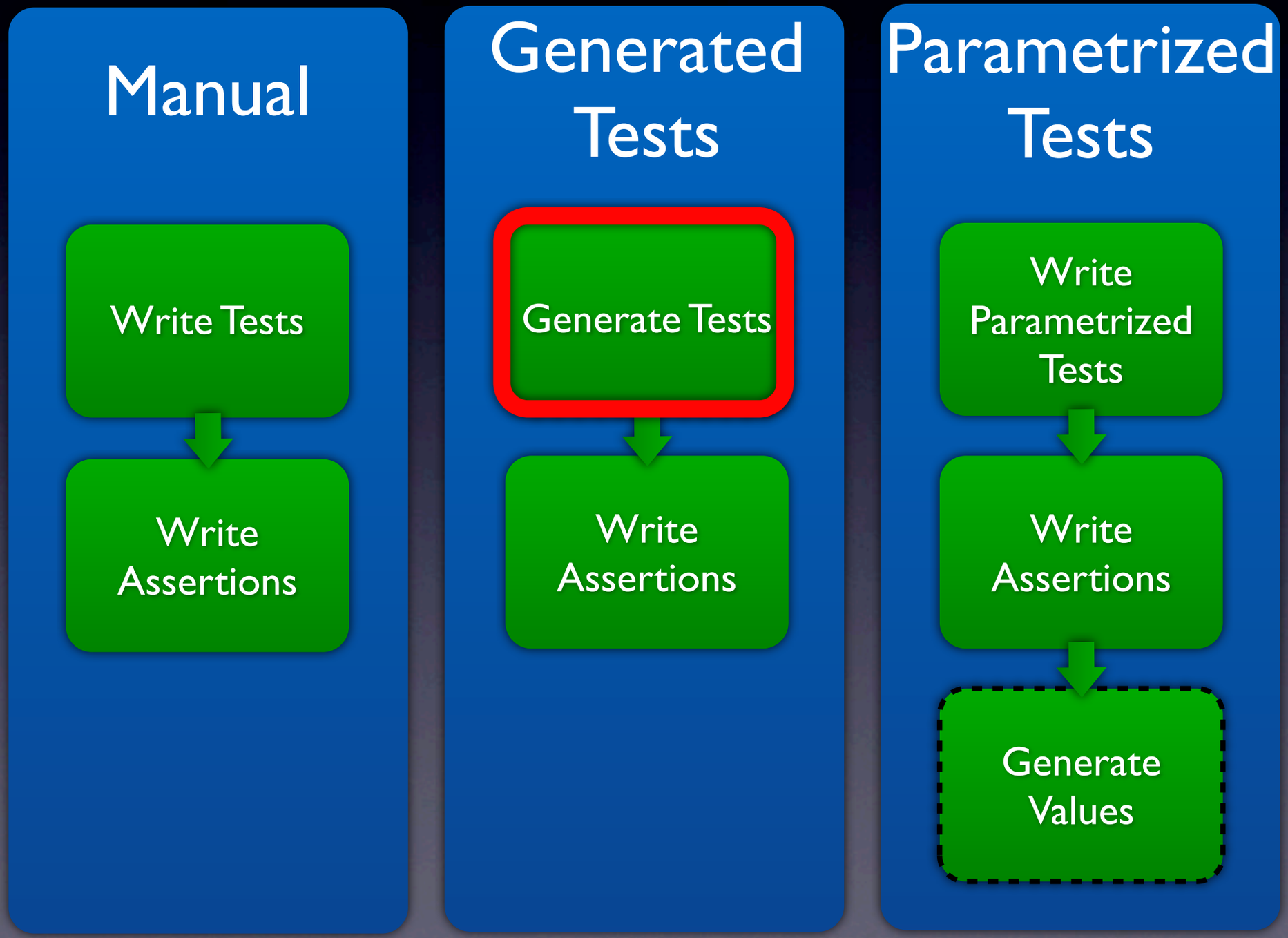
Mutation-based Generation of Tests and Oracles

Gordon Fraser and Andreas Zeller
Saarland University, Germany

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

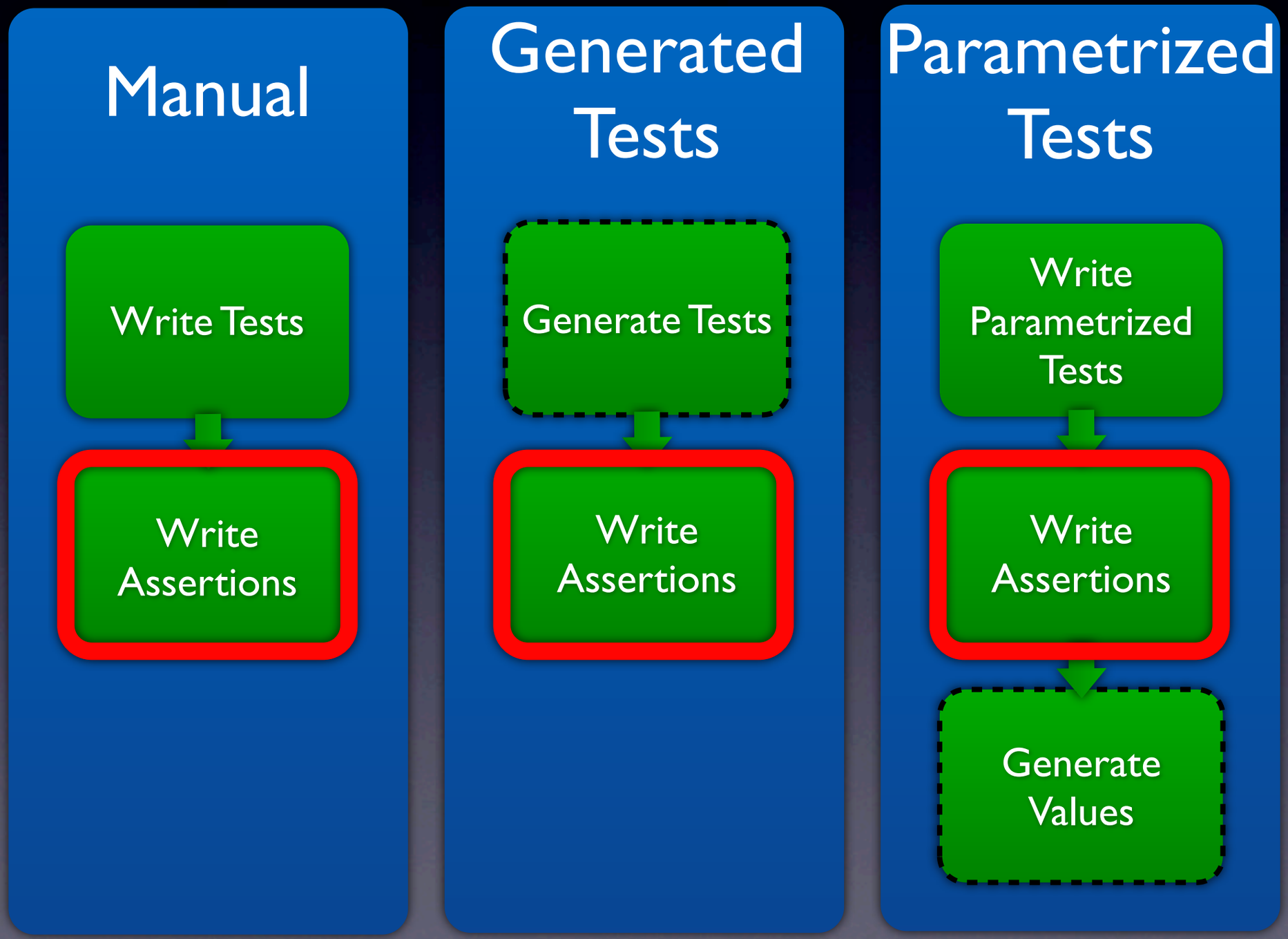


```
Instant var0 = GJChronology.DEFAULT_CUTOVER;
int var1 = 0;
FixedDateTimeZone var2 = null;
try {
  var2 = new org.joda.time.tz.FixedDateTimeZone(var0, var0, var1, var1);
} catch(IllegalArgumentException e) {}
DateTime var3 = new org.joda.time.DateTime(var0, var2);
Property var4 = var3.secondOfDay();
DateTime var5 = new org.joda.time.DateTime(var4, var4);
DateTimeComparator var6 = org.joda.time.DateTimeComparator.getInstance(var5, var5);
DateTimeFieldType var7 = var6.getUpperLimit();
DateTime var8 = null;
try {
  var8 = var5.withMonthOfYear(var1);
} catch(IllegalFieldValueException e) {}
Property var9 = null;
try {
  var9 = var5.property(var7);
} catch(IllegalArgumentException e) {}
```

```
Instant var0 = GJChronology.DEFAULT_CUTOVER;
int var1 = 0;
FixedDateTimeZone var2 = null;
try {
  var2 = new
} catch(Illega
DateTime va
Property var
DateTime va
DateTimeCo                                          e(var5, var5);
DateTimeFie
DateTime va
try {
  var8 = var5
} catch(IllegalFieldValueException e) {}
Property var9 = null;
try {
  var9 = var5.property(var7);
} catch(IllegalArgumentException e) {}
```

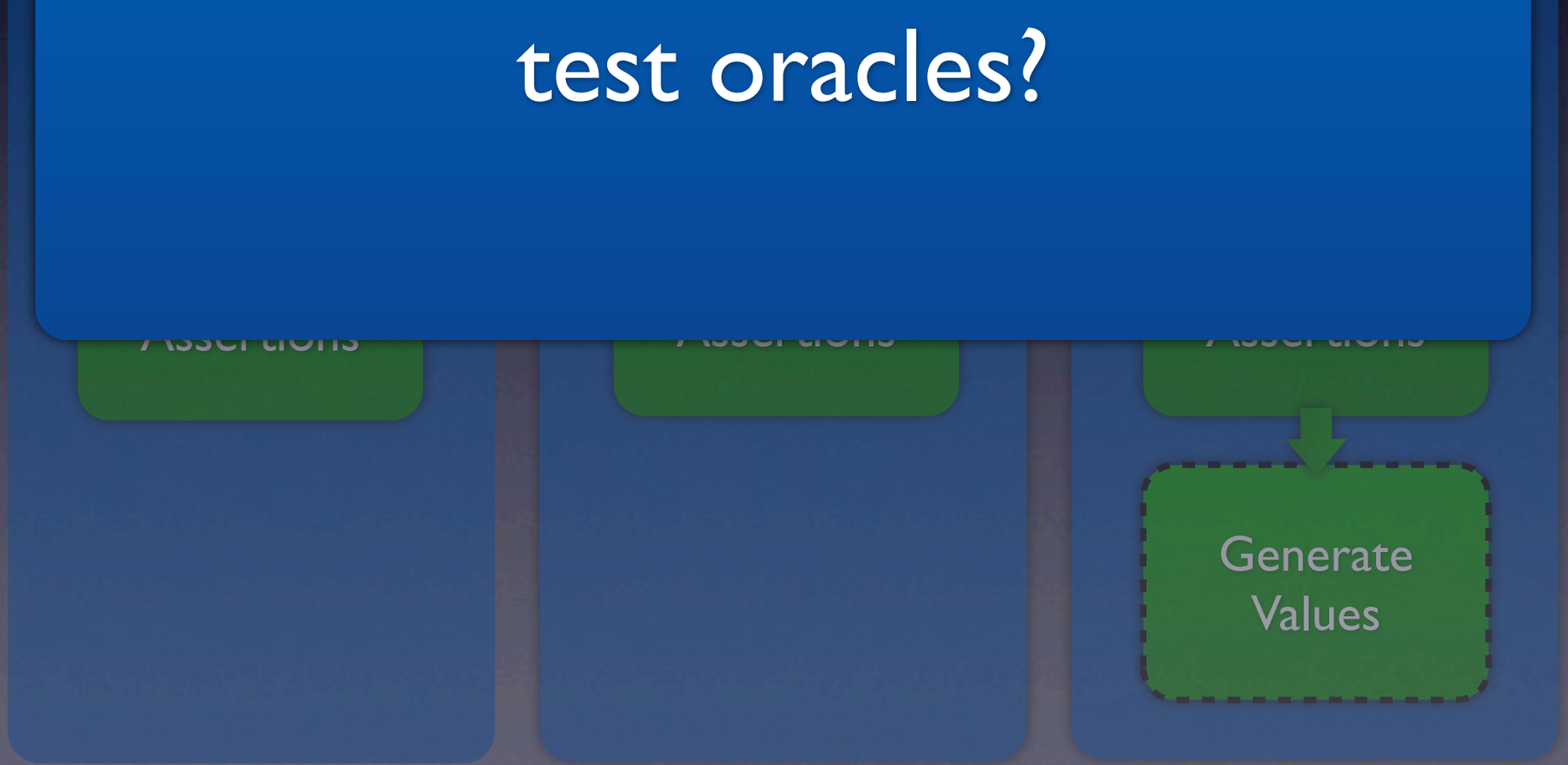
We need tests that *reveal faults* rather than covering code

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```



```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

How can we create test oracles?



```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.compareTo(date), 0);
date.plusYears(1);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getYear(), 2011);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
AR), 7);
H), 15);

How can we create *effective* test oracles?

assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);

μTest

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return 2 + x;
  }
}
```

```
void test() {
  f = new Foo();
  y = f.bar(10);
  assert(y==20);
}
```

```
void test() {
  f = new Foo();
  y = f.bar(10);
  assert(y==20);
}
```

Reaching Mutants

- Approach level
- Branch distance
- Necessity conditions

Propagating Mutants

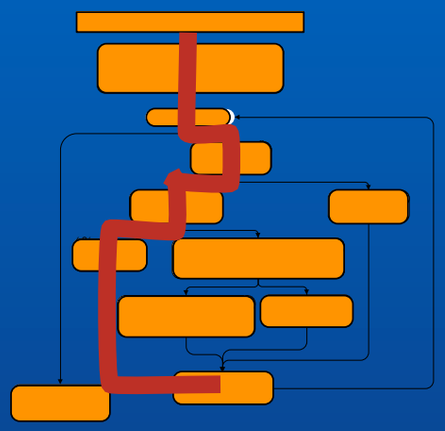
Constructor

Methodcall

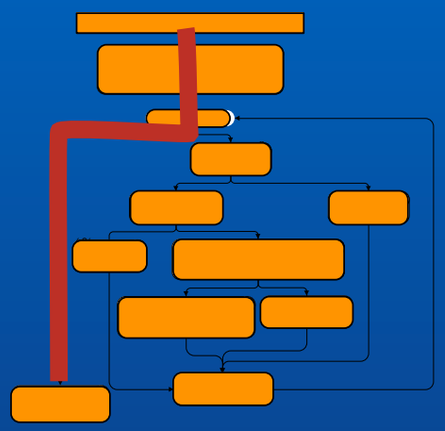
Methodcall

Methodcall

Program trace



Mutant trace



Impact:

Σ Control flow or data changes

μTest

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return 2 + x;
  }
}
```

```
void test() {
  f = new Foo();
  y = f.bar(10);
  assert(y==20);
}
```

Regular Execution

Constructor

Methodcall

Methodcall

Methodcall

Regular Execution

Constructor

Observation 1

Observation 2

Methodcall

Observation 1

Observation 2

Methodcall

Observation 1

Observation 2

Methodcall

Observation 1

Observation 2

Regular Execution

Constructor

Methodcall

Methodcall

Methodcall

Observation 1 | Observation 2

Observation 1 | Observation 2

Observation 1 | Observation 2

Observation 1 | Observation 2

Mutant Execution

Constructor

Methodcall

Methodcall

Regular Execution

Constructor

Methodcall

Methodcall

Methodcall

Observation 1 | Observation 2

Observation 1 | Observation 2

Observation 1 | Observation 2

Observation 1 | Observation 2

Mutant Execution

Constructor

Methodcall

Methodcall

```
class Foo {
  int bar(int x) {
    return 2 * x;
  }
}
```

```
class Foo {
  int bar(int x) {
    return x * x;
  }
}
```

```
void test() {
  f = new Foo();
  y = f.bar(10);
}
```

assert ...

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2010);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getYear(), 2010);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

```
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

```
assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);
assertEquals(date.getValue(YEAR), 2011);
assertEquals(date.getValue(MONTH_OF_YEAR), 7);
assertEquals(date.getValue(DAY_OF_MONTH), 15);
assertEquals(date.getLocalMillis(), ...);
assertEquals(date, date);
assertEquals(date.compareTo(date), 0);
assertEquals(date.getYearOfEra(), ...);
assertEquals(date.getYearOfCentury(), ...);
assertEquals(date.getWeekyear(), ...);
assertEquals(date.getMonthOfYear(), 7);
assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);
```

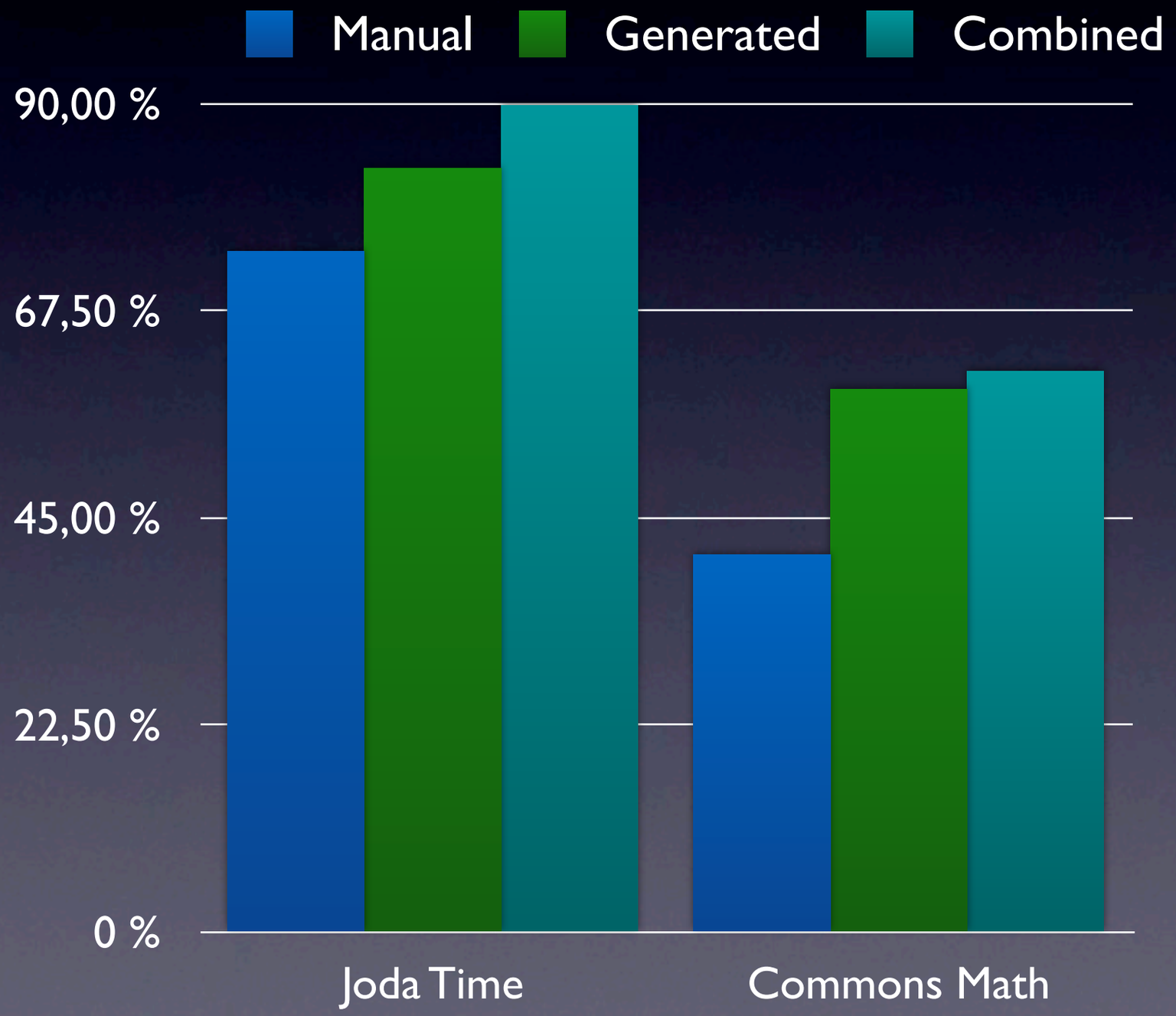
```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.getValue(YEAR), 2011);
```

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

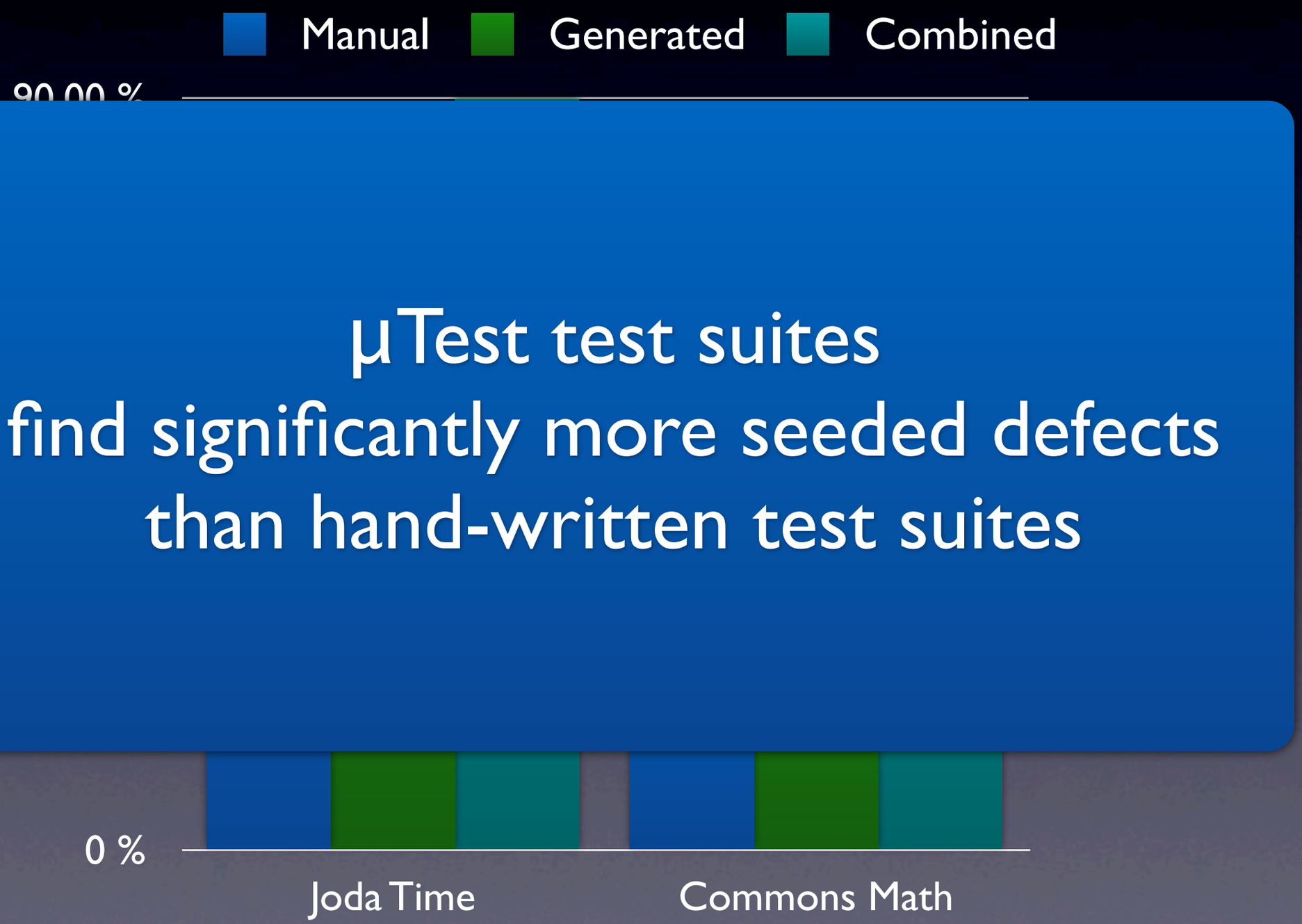
Evaluation: μTest

	Classes	Existing Tests	Generated Tests
Joda Time	123	3.493	1.268
Commons Math	220	1.739	2.706

Evaluation: μTest

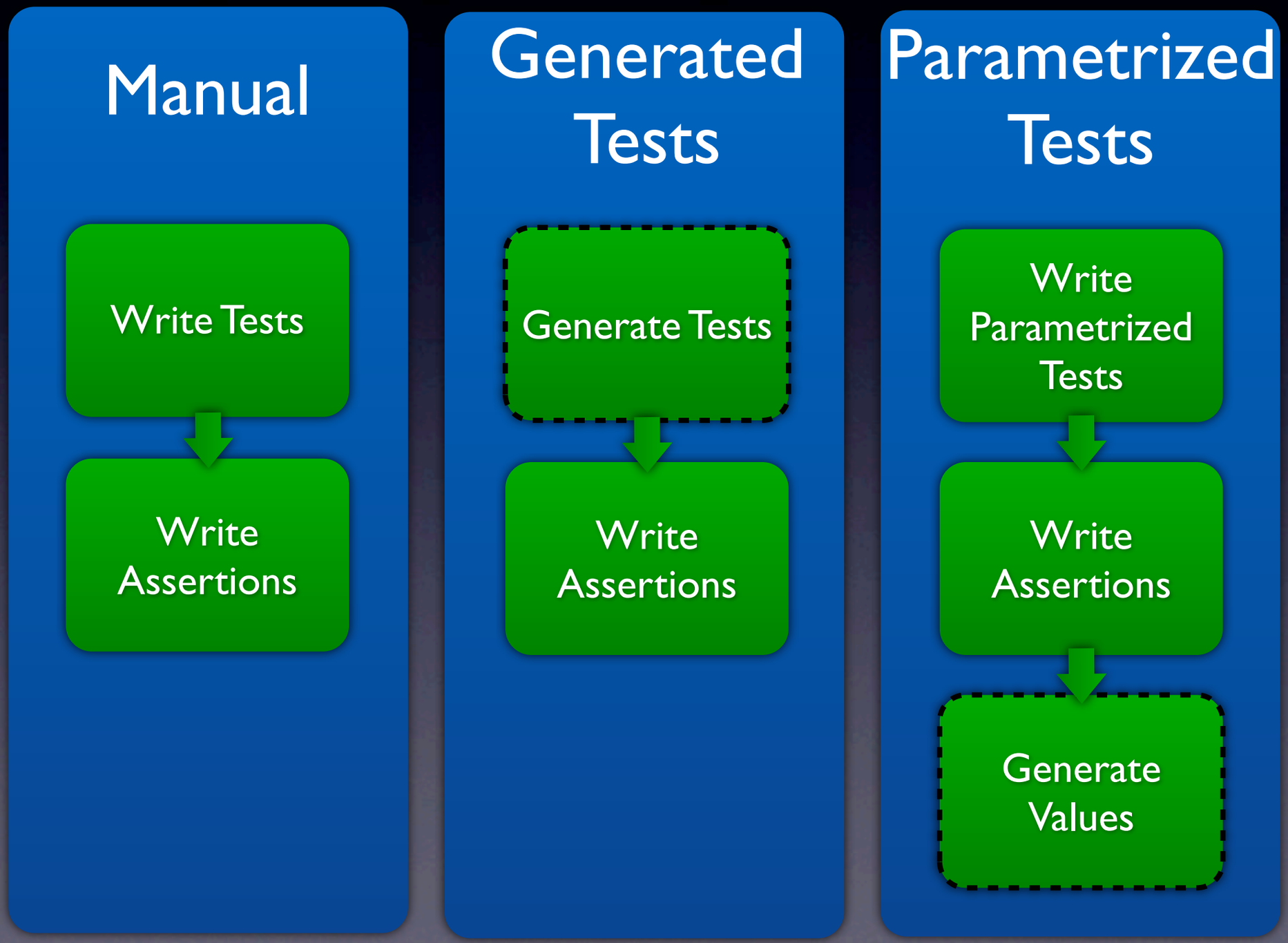


Evaluation: μTest

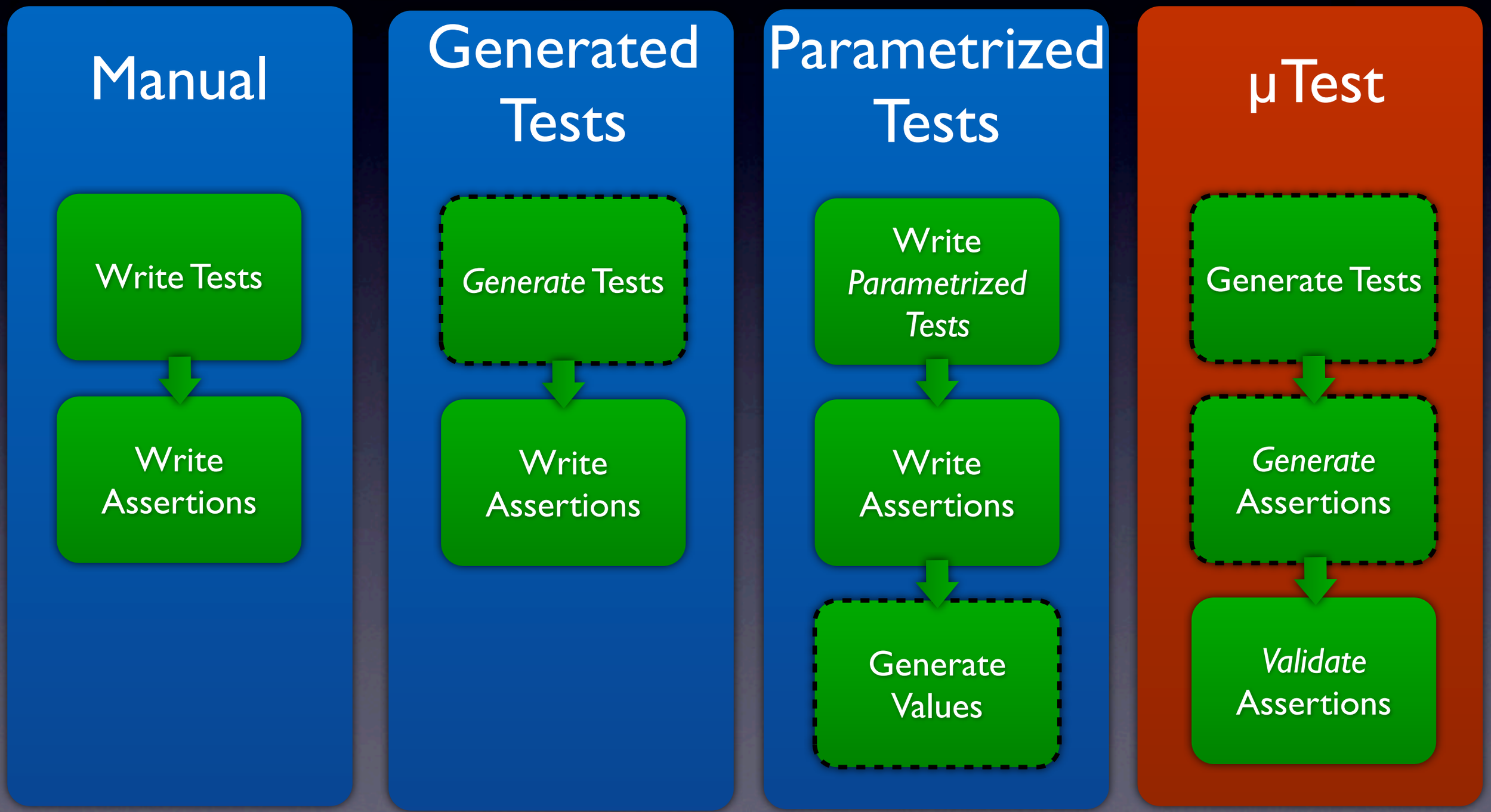


μTest test suites
find significantly more seeded defects
than hand-written test suites

```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```



```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```



assertEquals(date.getDayOfMonth(), ...);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
assertEquals(date.size(), 3);

We need tests that *reveal faults* rather than covering code

How can we create *effective* test oracles?

assertEquals(date.getWeekOfWeekyear(), ...);
assertEquals(date.getDayOfWeek(), ...);
assertEquals(date.getDayOfMonth(), ...);

µTest

```
class Foo {
  int bar(int x) {
    return 2 * y;
  }
}
```

→

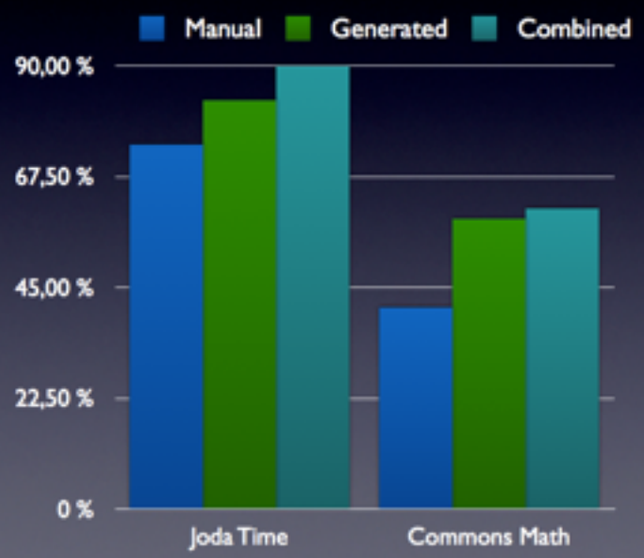
```
class Foo {
  int bar(int x) {
    return 2 + y;
  }
}
```

→

```
void test() {
  f = new Foo();
  y = f.bar(10);
  assert(y==20);
}
```

Summary

Evaluation




```
LocalDate date = new LocalDate(2010, 7, 15);
date.plusYears(1);
assertEquals(date.getYear(), 2011);
```

Manual	Generated Tests	Parametrized Tests	µTest
Write Tests	*Generate Tests*	Write *Parametrized Tests*	Generate Tests
Write Assertions	Write Assertions	Write Assertions	*Generate* Assertions
		Generate Values	*Validate* Assertions