The University Of Sheffield.

COMBINING MULTIPLE COVERAGE CRITERIA IN SEARCH-BASED UNIT TEST GENERATION

SSBSE 2015

José Miguel Rojas

j.rojas@sheffield.ac.uk

Joint work with José Campos, Mattia Vivanti, Gordon Fraser and Andrea Arcuri

AUTOMATED UNIT TEST GENERATION

Class under Test

Java - comparators/src/java/collections/comparators/FixedOrderComparator.java - Eclipse - /Users/jmr/Papers/evosuite-stu...

Java | Debug

Packag | JUnit

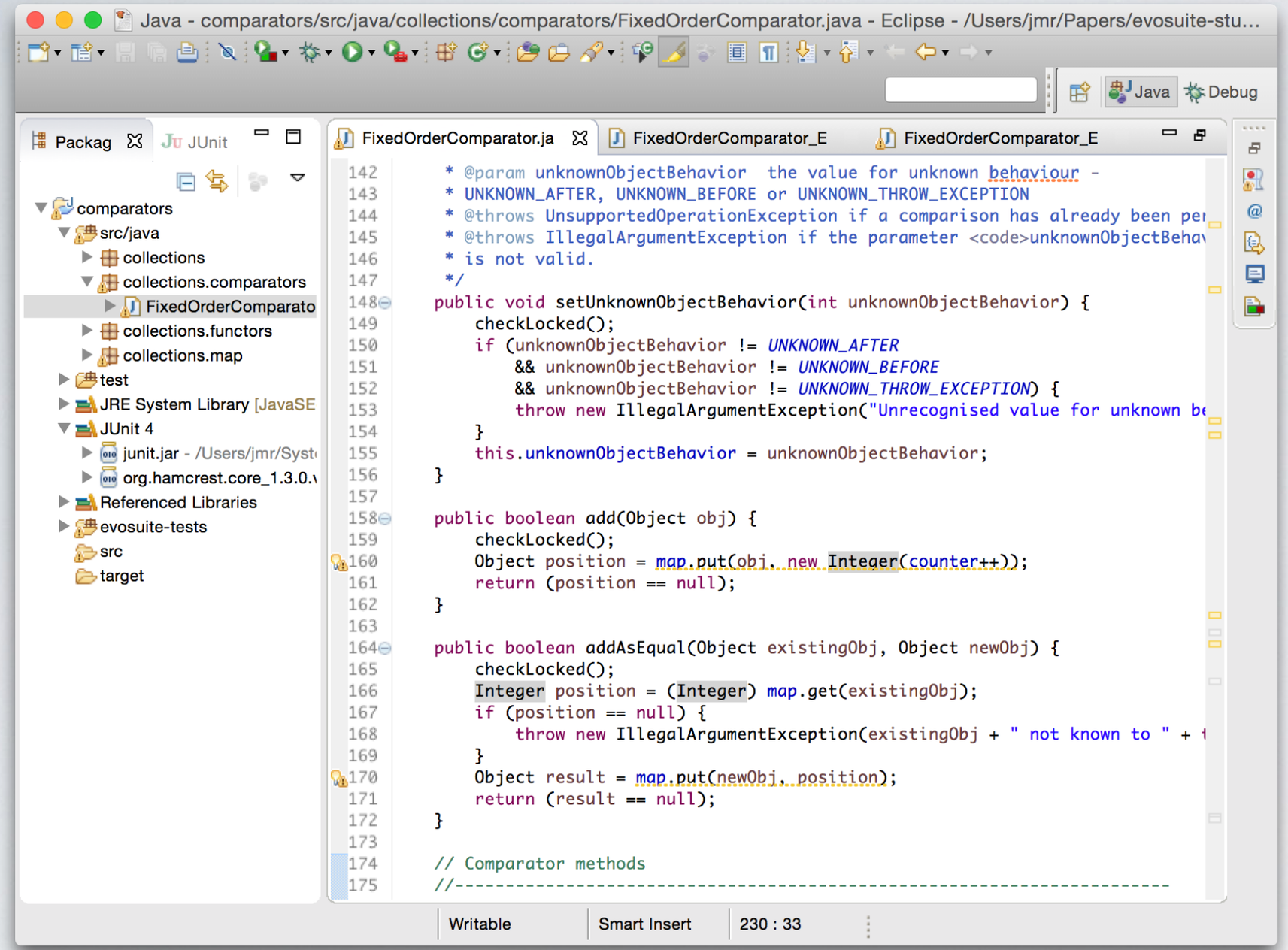
- comparators
  - src/java
    - collections
    - collections.comparators
      - FixedOrderComparato
    - collections.functors
    - collections.map
  - test
  - JRE System Library [JavaSE
  - JUnit 4
    - junit.jar - /Users/jmr/Syst
    - org.hamcrest.core_1.3.0.v
  - Referenced Libraries
  - evosuite-tests
  - src
  - target

FixedOrderComparator.ja | FixedOrderComparator_E | FixedOrderComparator_E

```
142      * @param unknownObjectBehavior  the value for unknown behaviour -
143      * UNKNOWN_AFTER, UNKNOWN_BEFORE or UNKNOWN_THROW_EXCEPTION
144      * @throws UnsupportedOperationException if a comparison has already been per
145      * @throws IllegalArgumentException if the parameter <code>unknownObjectBehav
146      * is not valid.
147      */
148     public void setUnknownObjectBehavior(int unknownObjectBehavior) {
149         checkLocked();
150         if (unknownObjectBehavior != UNKNOWN_AFTER
151             && unknownObjectBehavior != UNKNOWN_BEFORE
152             && unknownObjectBehavior != UNKNOWN_THROW_EXCEPTION) {
153             throw new IllegalArgumentException("Unrecognised value for unknown be
154         }
155         this.unknownObjectBehavior = unknownObjectBehavior;
156     }
157
158     public boolean add(Object obj) {
159         checkLocked();
160         Object position = map.put(obj, new Integer(counter++));
161         return (position == null);
162     }
163
164     public boolean addAsEqual(Object existingObj, Object newObj) {
165         checkLocked();
166         Integer position = (Integer) map.get(existingObj);
167         if (position == null) {
168             throw new IllegalArgumentException(existingObj + " not known to " + t
169         }
170         Object result = map.put(newObj, position);
171         return (result == null);
172     }
173
174     // Comparator methods
175     //-------------------------------------------------------------------------
```

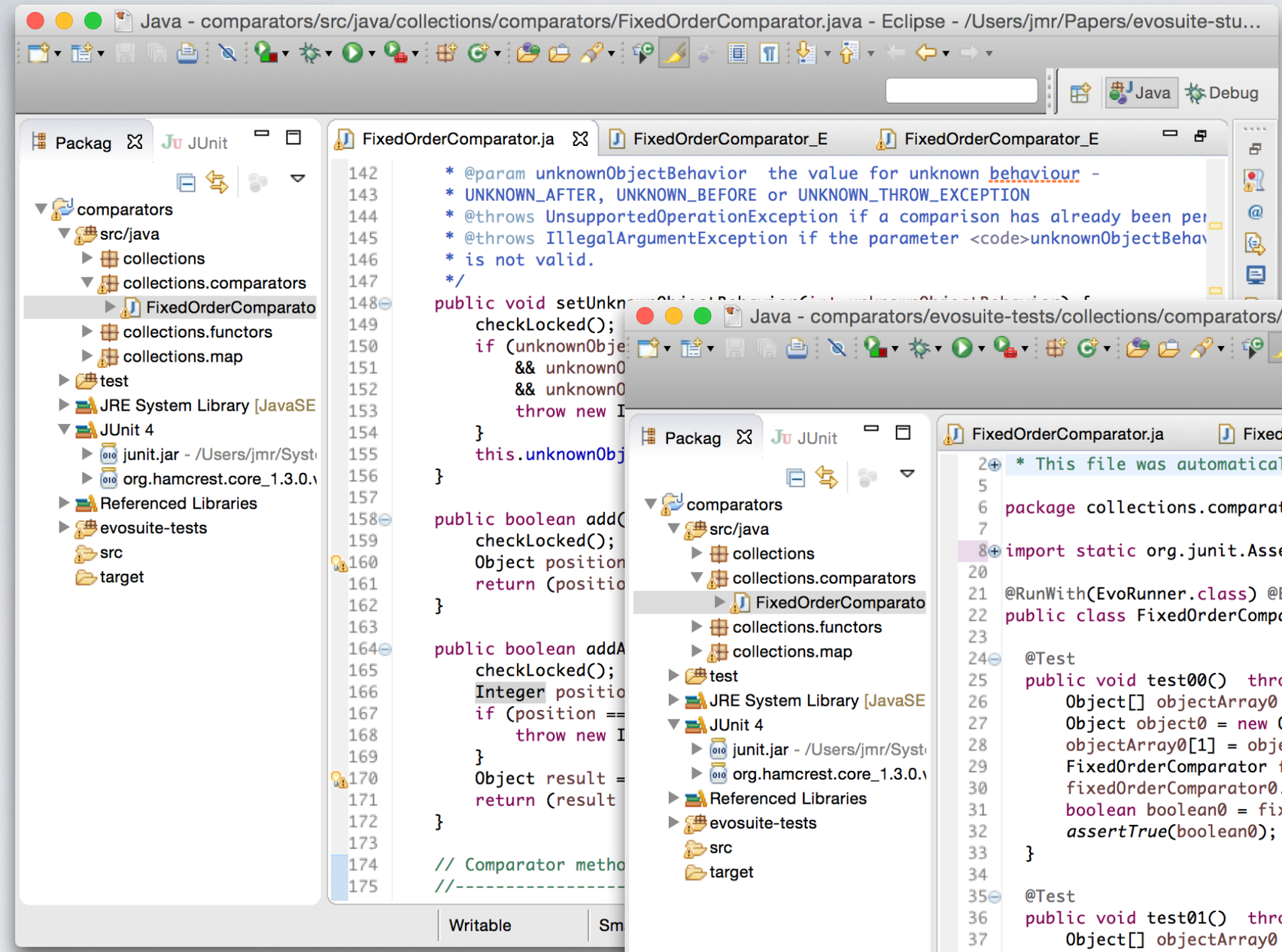
Writable | Smart Insert | 230 : 33

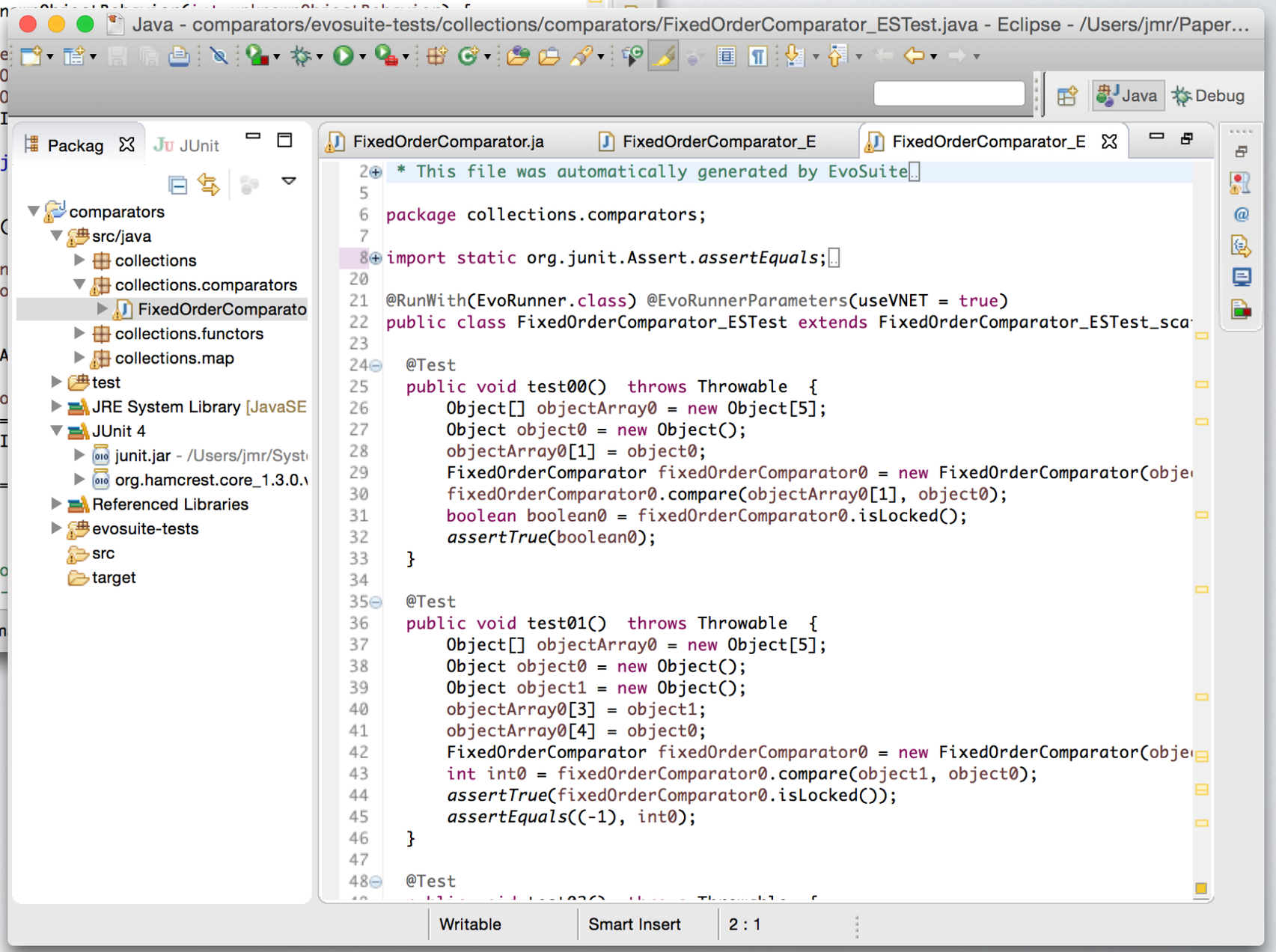
Class under Test



Automated Unit Test Generation

Class under Test

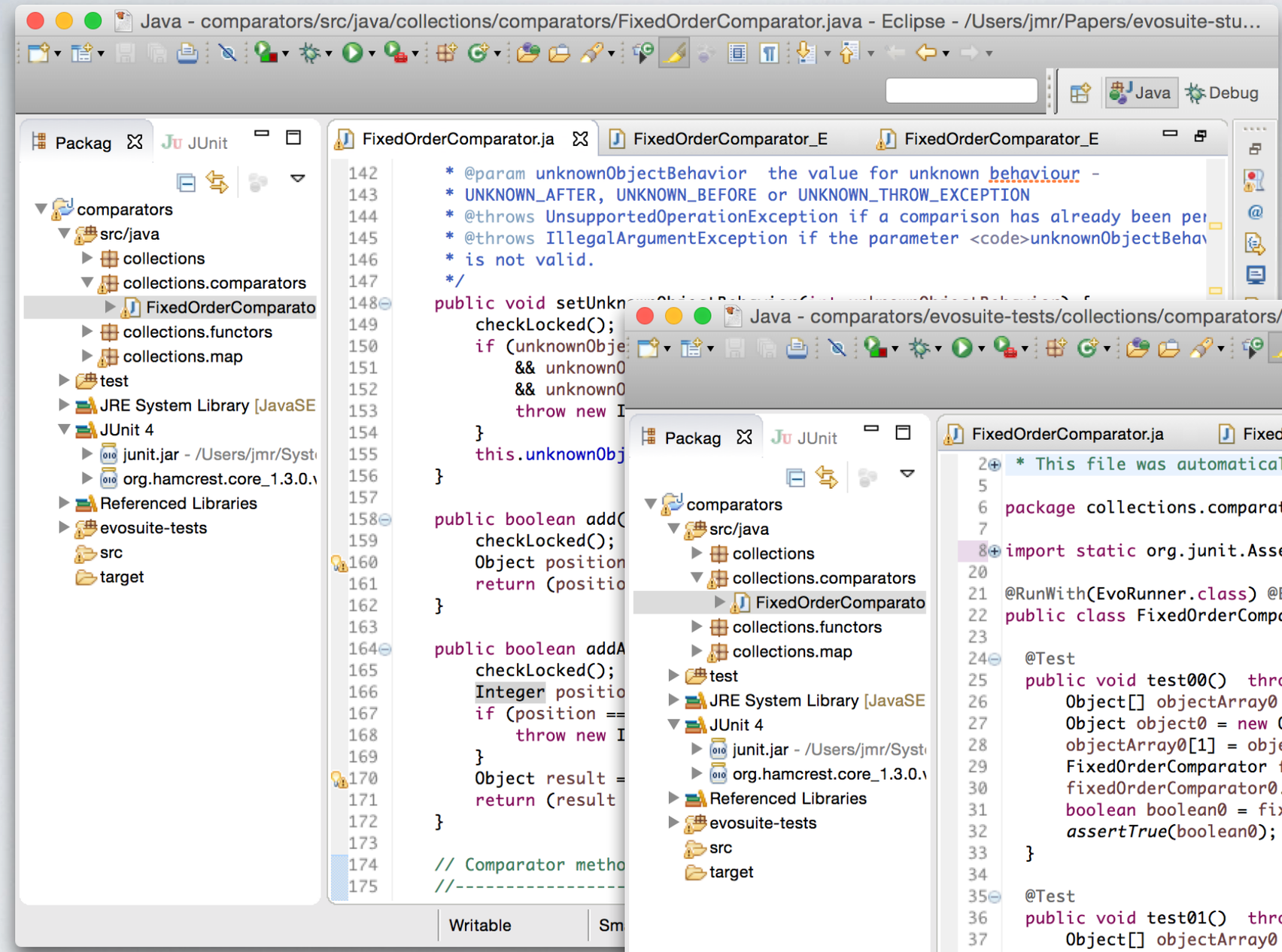




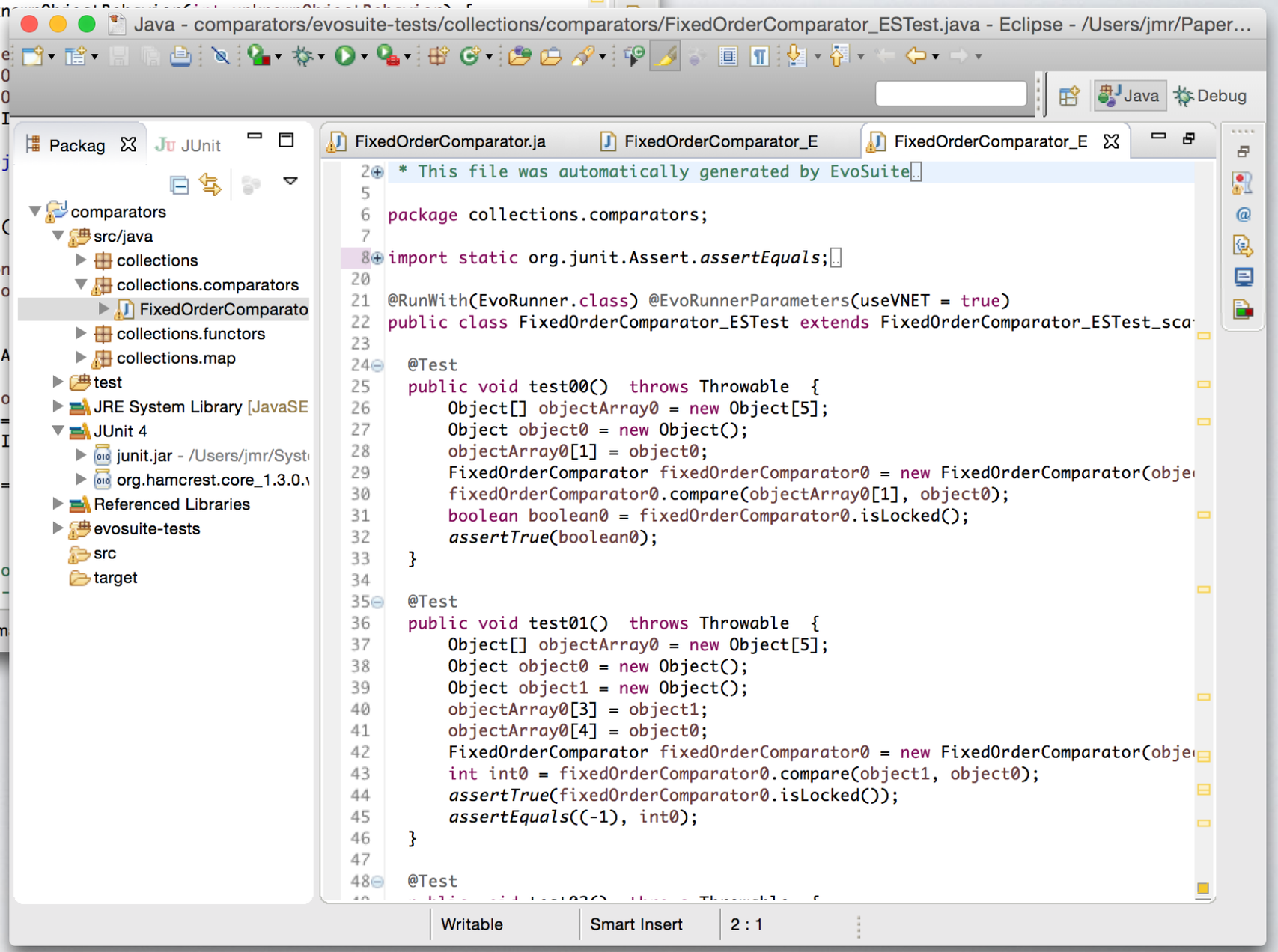
Automated Unit Test Generation

Test Suite

Class under Test



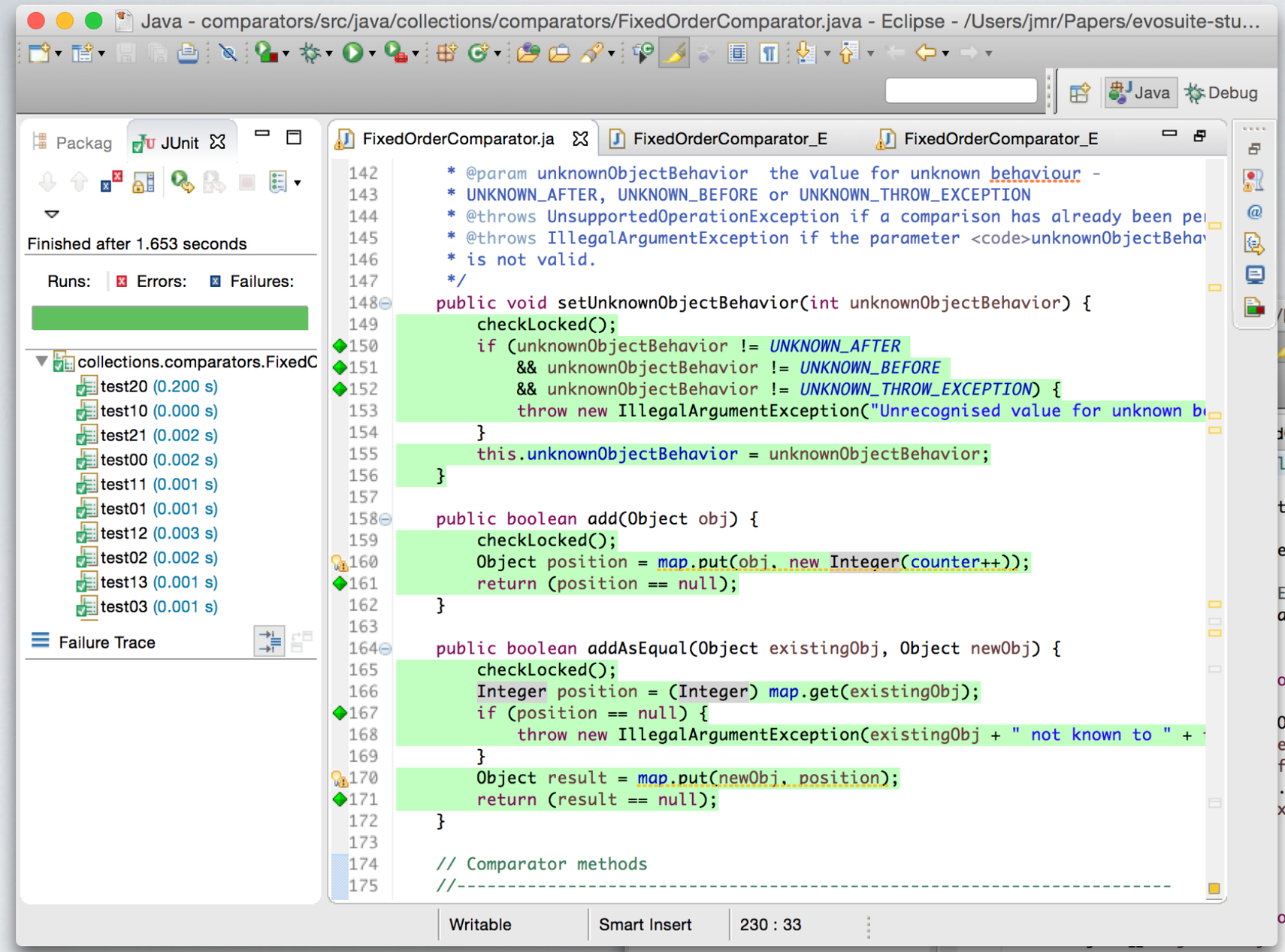
Optimised for Branch Coverage



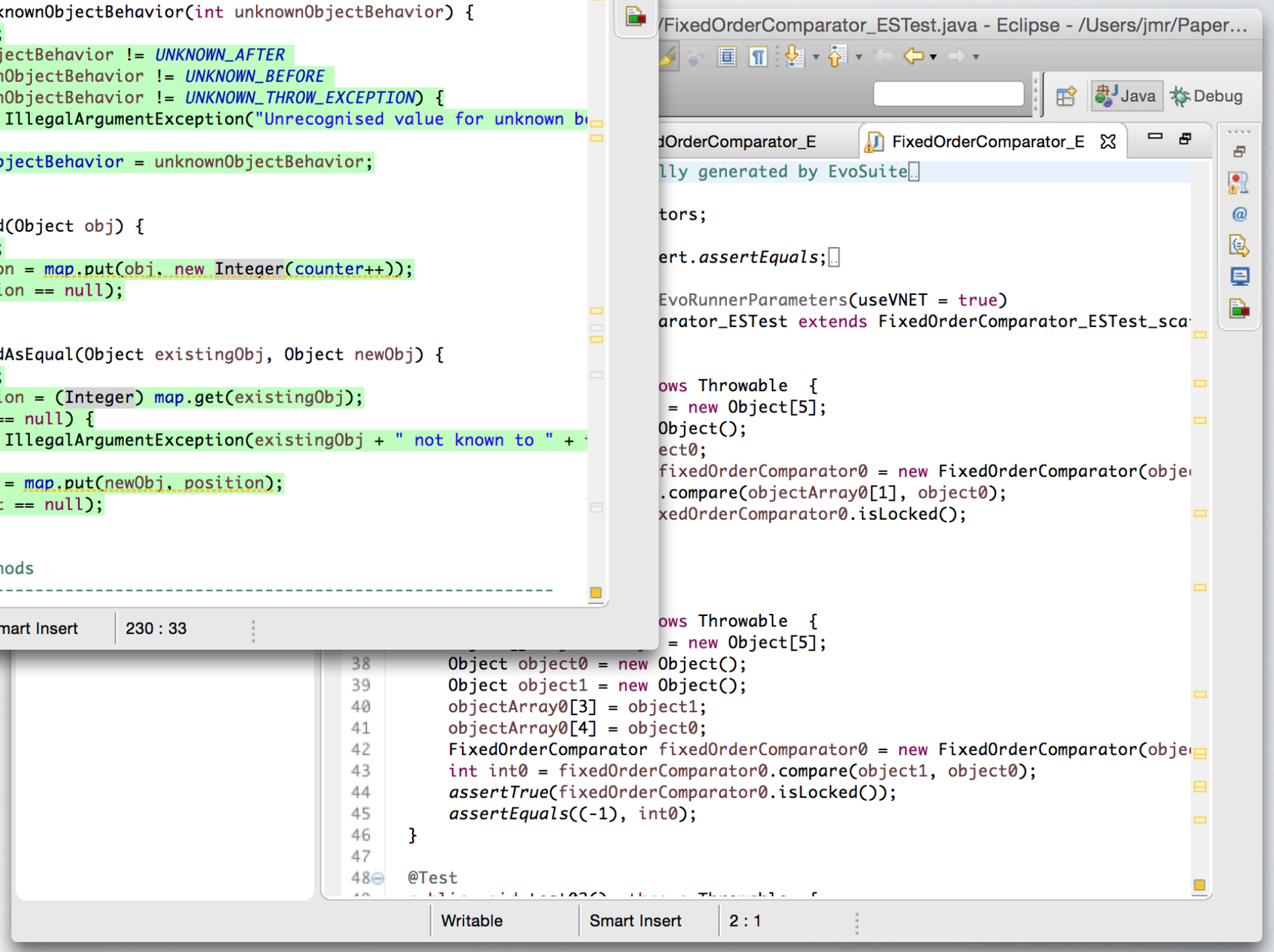
Automated Unit Test Generation

Test Suite

Class under Test



Optimised for Branch Coverage



Automated Unit Test Generation

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class ArrayIntList
public int set(int idx,
               int element){
  checkRange(idx);
  incrModCount();
  int oldval = _data[idx];
  _data[idx] = element;
  return oldval;
}
```

Class Under Test

```
@Test
void test9(){
  ArrayIntList l =
        new ArrayIntList();
  // Undeclared exception!
  try {
    int int0 = l.set(20, 20);
    fail("Expecting IOOBExc");
  } catch(IOOBExc e) {
    // Should be at least 0
    // and less than 0,
    // found 20
  }
}
```

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class ArrayIntList
public int set(int idx,
               int element){
   checkRange(idx);
   incrModCount();
   int oldval = _data[idx];
   _data[idx] = element;
   return oldval;
}
```

Class Under Test

```
@Test
void test9(){
   ArrayIntList l =
         new ArrayIntList();
   // Undeclared exception!
   try {
      int int0 = l.set(20, 20);
      fail("Expecting IOOBExc");
   } catch(IOOBExc e) {
      // Should be at least 0
      // and less than 0,
      // found 20
   }
}
```

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

```
// Class Complex
public Complex log(){
  if (isNaN) return NaN;
  double r = log(abs());
  double i;
  i = atan2(imaginary, real);
  return createComplex(r, i);
}
public Complex pow(double x){
  Complex c = this.log();
  return c.multiply(x).exp();
}
```

Class Under Test

```
@Test
void test1() {
  Complex c0 = new Complex(NaN);
  Complex c1 = c0.pow(NaN);
  double d = c1.getArgument();
  assertEquals(NaN, d, 0.01);
}
@Test
void test2() {
  Complex c0 = Complex.ZERO;
  Complex c1 = c0.pow(c0);
  assertFalse(c1.isInfinite());
  assertTrue(c1.isNaN());
}
```

Test Suite

IS 100% BRANCH COVERAGE ENOUGH?

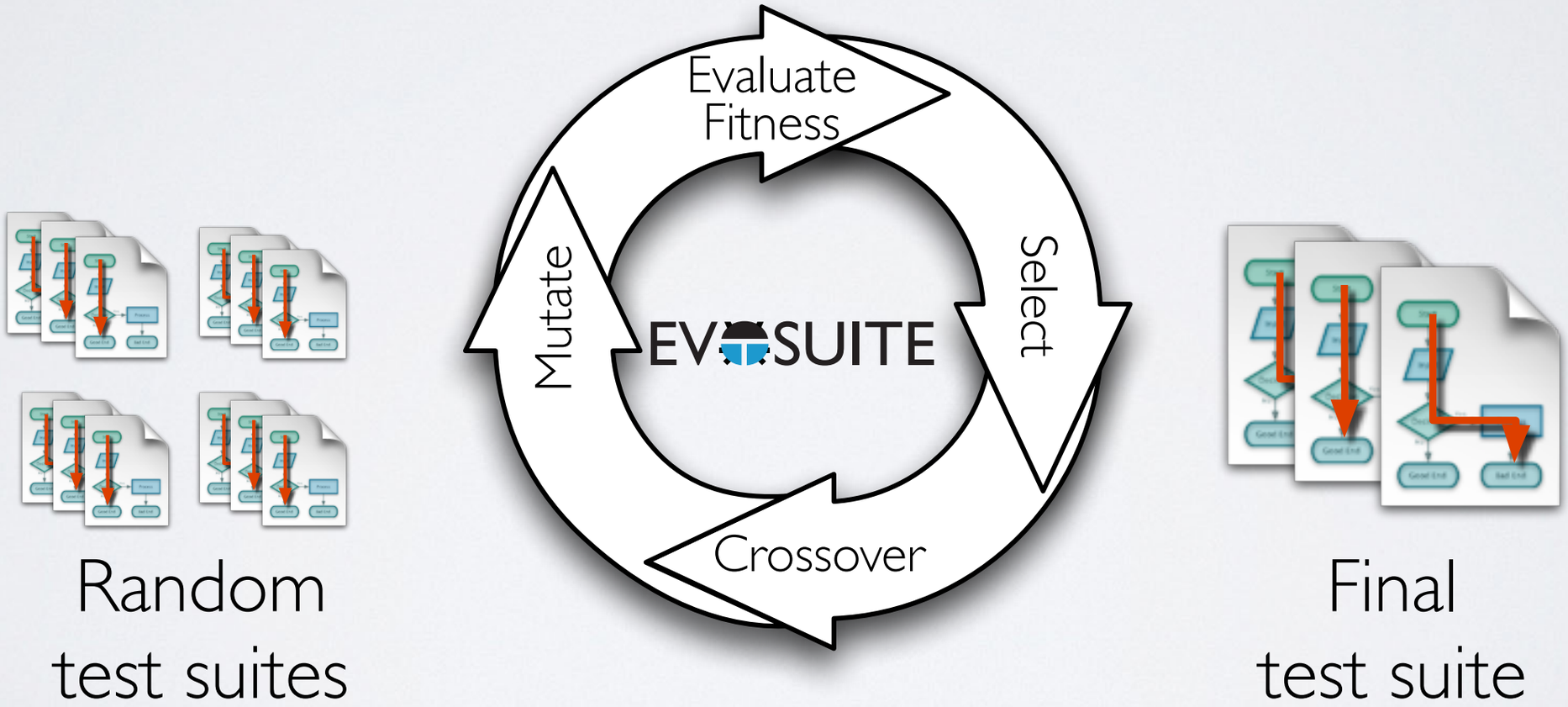
```
// Class Complex
public Complex log(){
   if (isNaN) return NaN;
   double r = log(abs());
   double i;
   i = atan2(imaginary, real);
   return createComplex(r, i);
}
public Complex pow(double x){
   Complex c = this.log();
   return c.multiply(x).exp();
}
```

Class Under Test

```
@Test
void test1() {
   Complex c0 = new Complex(NaN);
   Complex c1 = c0.pow(NaN);
   double d = c1.getArgument();
   assertEquals(NaN, d, 0.01);
}
@Test
void test2() {
   Complex c0 = Complex.ZERO;
   Complex c1 = c0.pow(c0);
   assertFalse(c1.isInfinite());
   assertTrue(c1.isNaN());
}
```

Test Suite

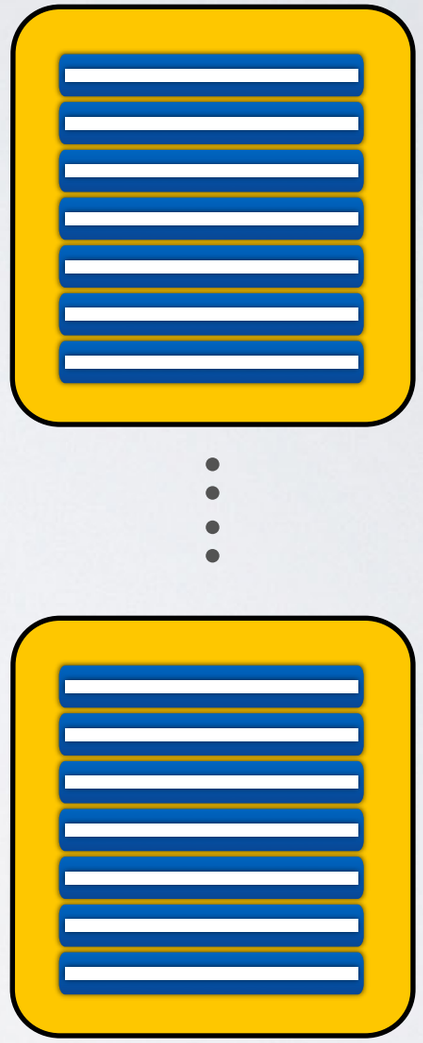
WHOLE TEST SUITE GENERATION



Fraser and Arcuri, Whole test suite generation. TSE. 2013.

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

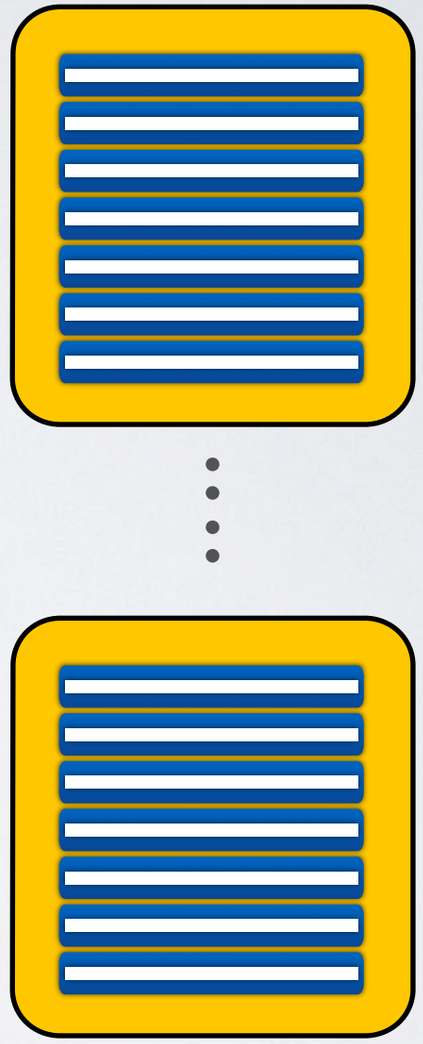
```
// Class Complex
 public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

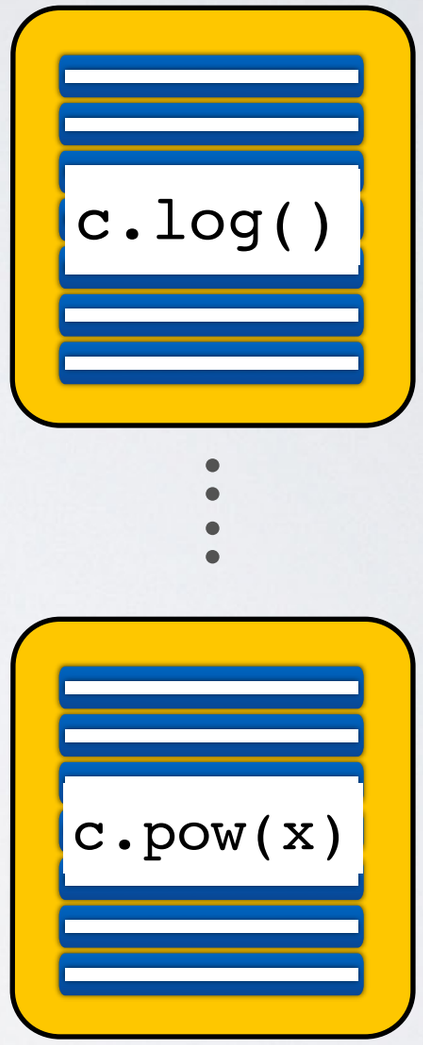
```
// Class Complex
public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Method Coverage

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

```
// Class Complex
public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```

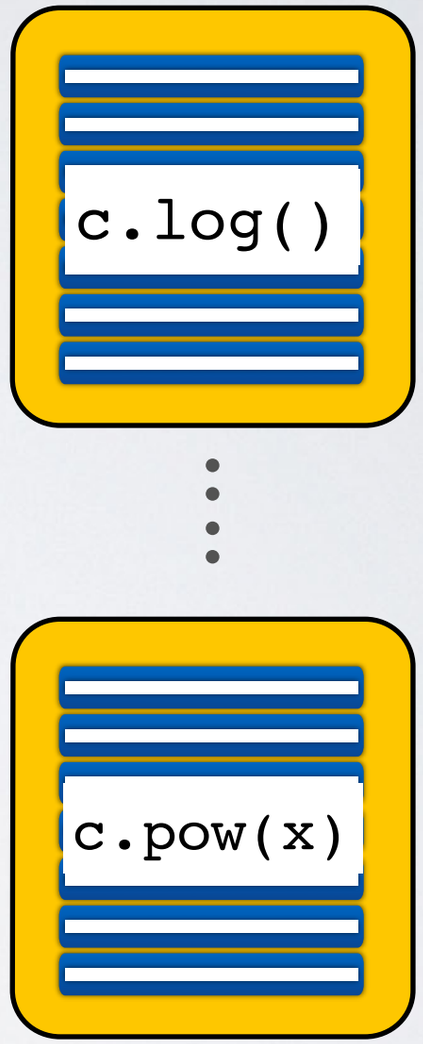


Top-level Method Coverage

FITNESS FUNCTIONS

(I.E., COVERAGE CRITERIA)

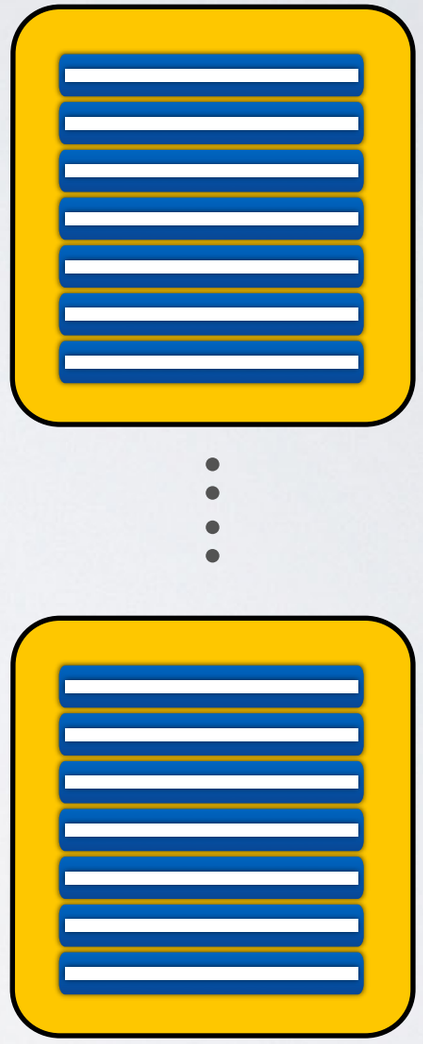
```
// Class Complex
public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Top-level Method Coverage No exception

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

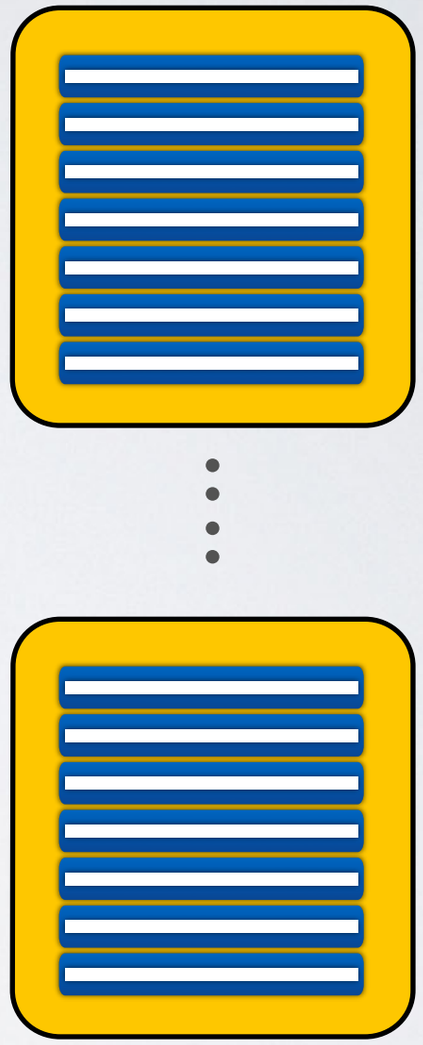
```
// Class Complex
 public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Line Coverage

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

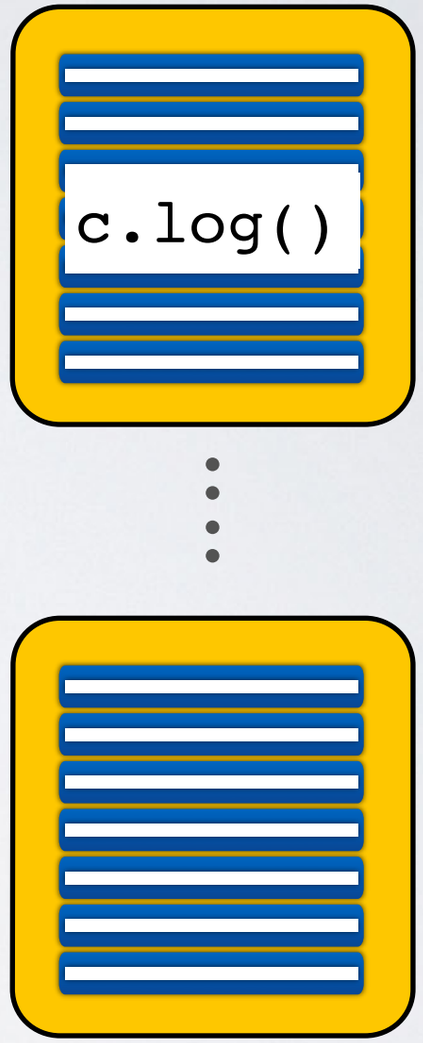
```
// Class Complex
 public Complex log(){
1  if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Branch Coverage

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

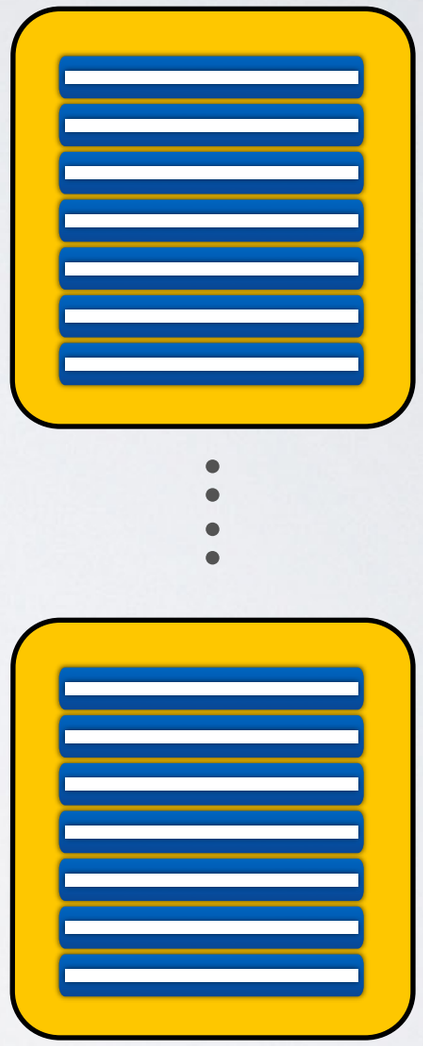
```
// Class Complex
 public Complex log(){
1  if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Direct Branch Coverage

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

```
// Class Complex
 public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



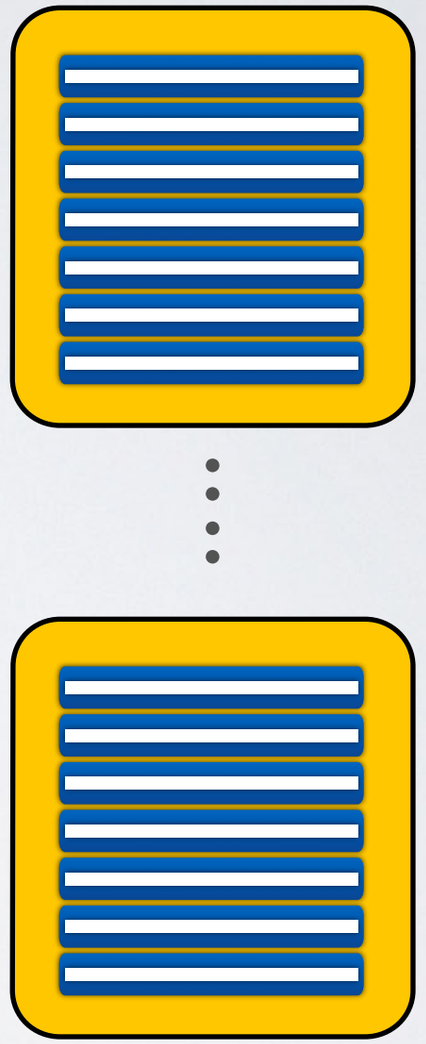
Weak Mutation

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

```
// Class Complex
 public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```

- -
- 0
- +

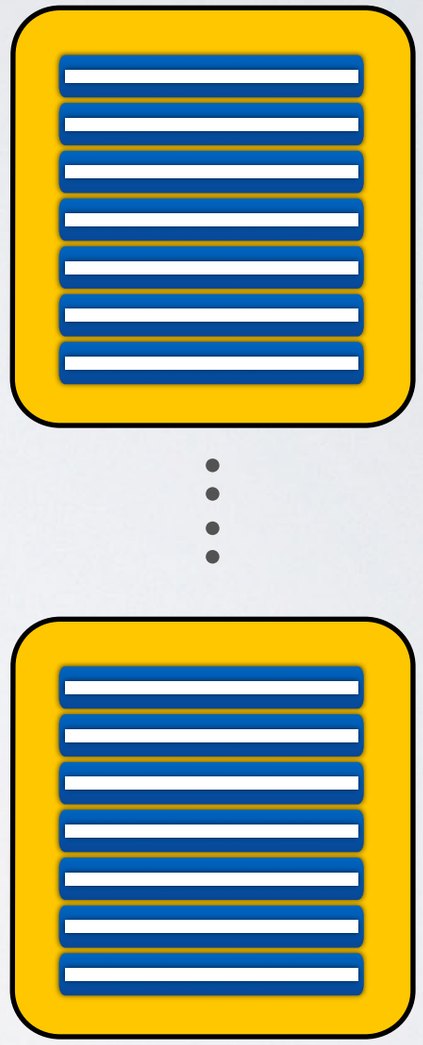
- null
- nonnull



Output Coverage
(e.g., Alshahwan and Harman, ISSTA'14)

FITNESS FUNCTIONS
(I.E., COVERAGE CRITERIA)

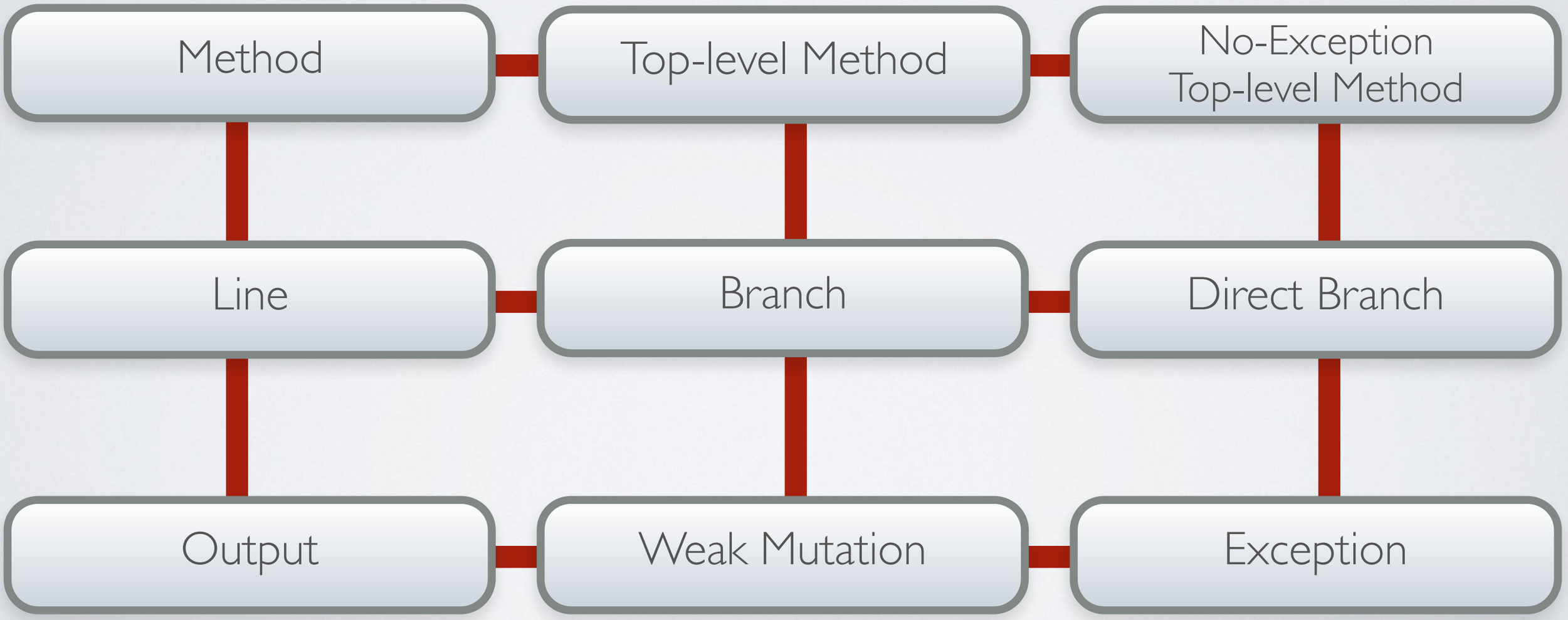
```
// Class Complex
 public Complex log(){
1   if (isNaN) return NaN;
2   double r = log(abs());
3   double i;
4   i = atan2(imaginary, real);
5   return createComplex(r, i);
}
 public Complex pow(double x)
        throws NullArgumentException{
6   Complex c = this.log();
7   return c.multiply(x).exp();
}
```



Number of exceptions

FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA



FITNESS FUNCTIONS

COMBINING MULTIPLE CRITERIA

Method

Top-level Method

No-Exception Top-level Method

$$fitness(\text{Suite}) = \sum_{i=1}^{n} w_i \times f_i$$

nch

Output

Weak Mutation

Exception

RESEARCH QUESTIONS

RESEARCH QUESTIONS

- **RQ1**. What are the effects of adding a second coverage criterion on test suite size and coverage?

RESEARCH QUESTIONS

- **RQ1**. What are the effects of adding a second coverage criterion on test suite size and coverage?
- **RQ2**. How does combining multiple coverage criteria influence the test suite size?

RESEARCH QUESTIONS

- **RQ1**. What are the effects of adding a second coverage criterion on test suite size and coverage?

- **RQ2**. How does combining multiple coverage criteria influence the test suite size?

- **RQ3**. How does combining multiple coverage criteria affect the performance of the constituent criteria?

RESEARCH QUESTIONS

- **RQ1**. What are the effects of adding a second coverage criterion on test suite size and coverage?
- **RQ2**. How does combining multiple coverage criteria influence the test suite size?
- **RQ3**. How does combining multiple coverage criteria affect the performance of the constituent criteria?
- **RQ4**. How does coverage vary with increasing search budget?

EXPERIMENTAL SETUP

EXPERIMENTAL SETUP

EVOSUITE

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects

+20,000 Classes

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects

+20,000 Classes

650

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects

+20,000 Classes

650

10

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects
+20,000 Classes

650

10

2/10

EXPERIMENTAL SETUP

EVOSUITE

SF110 Corpus

110 Projects
+20,000 Classes

650

10

2/10

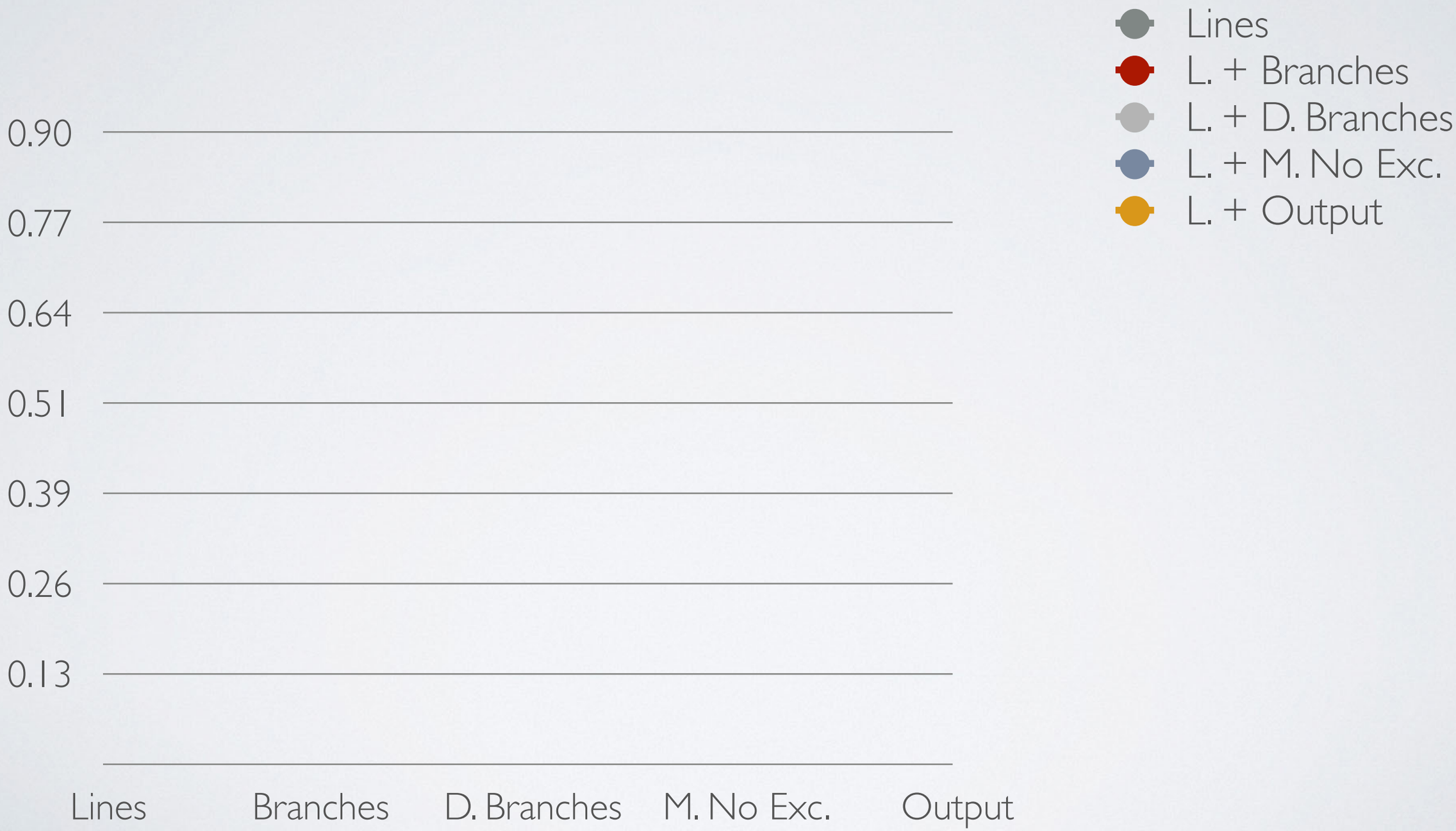
40/5

TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion

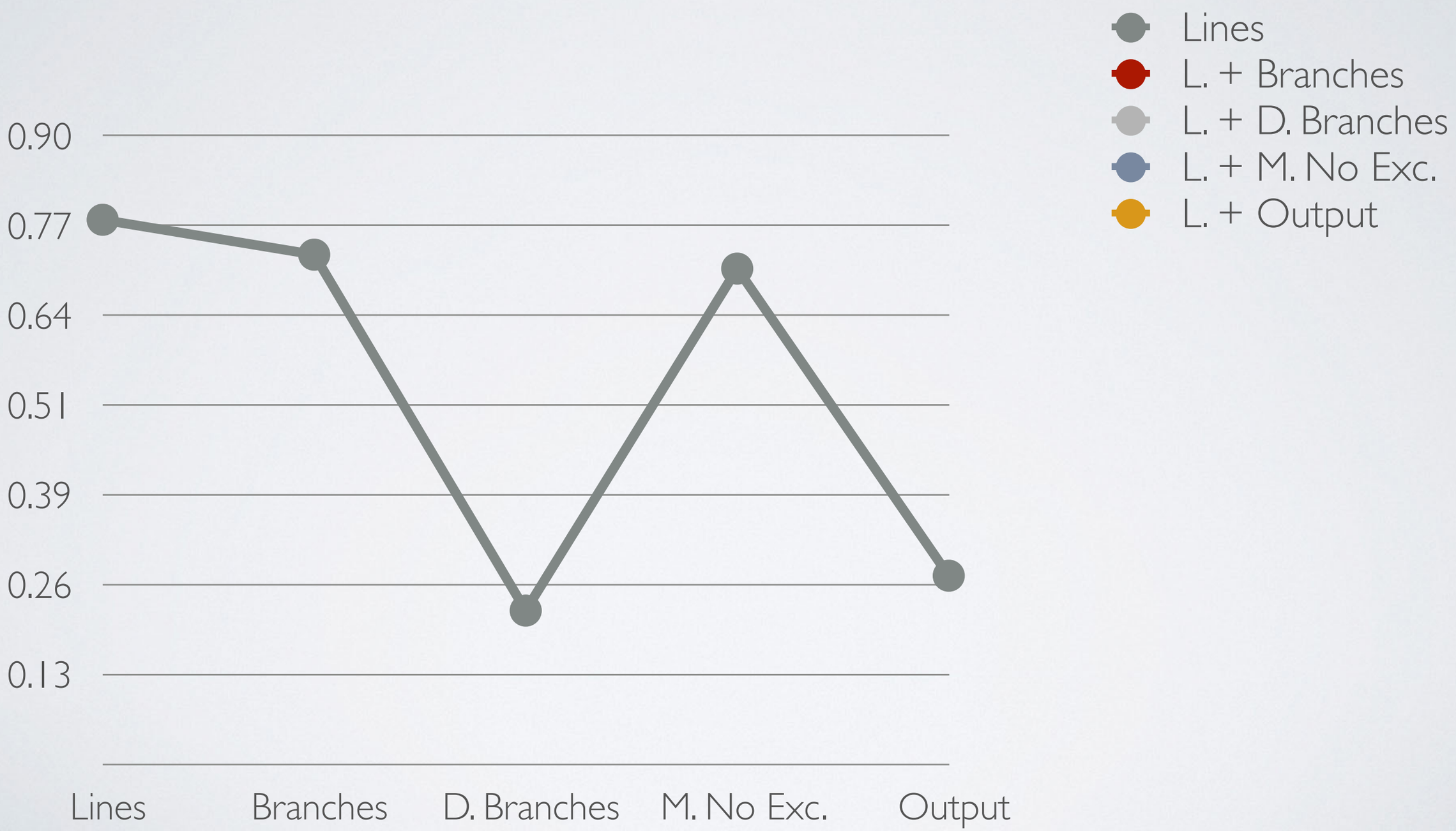
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



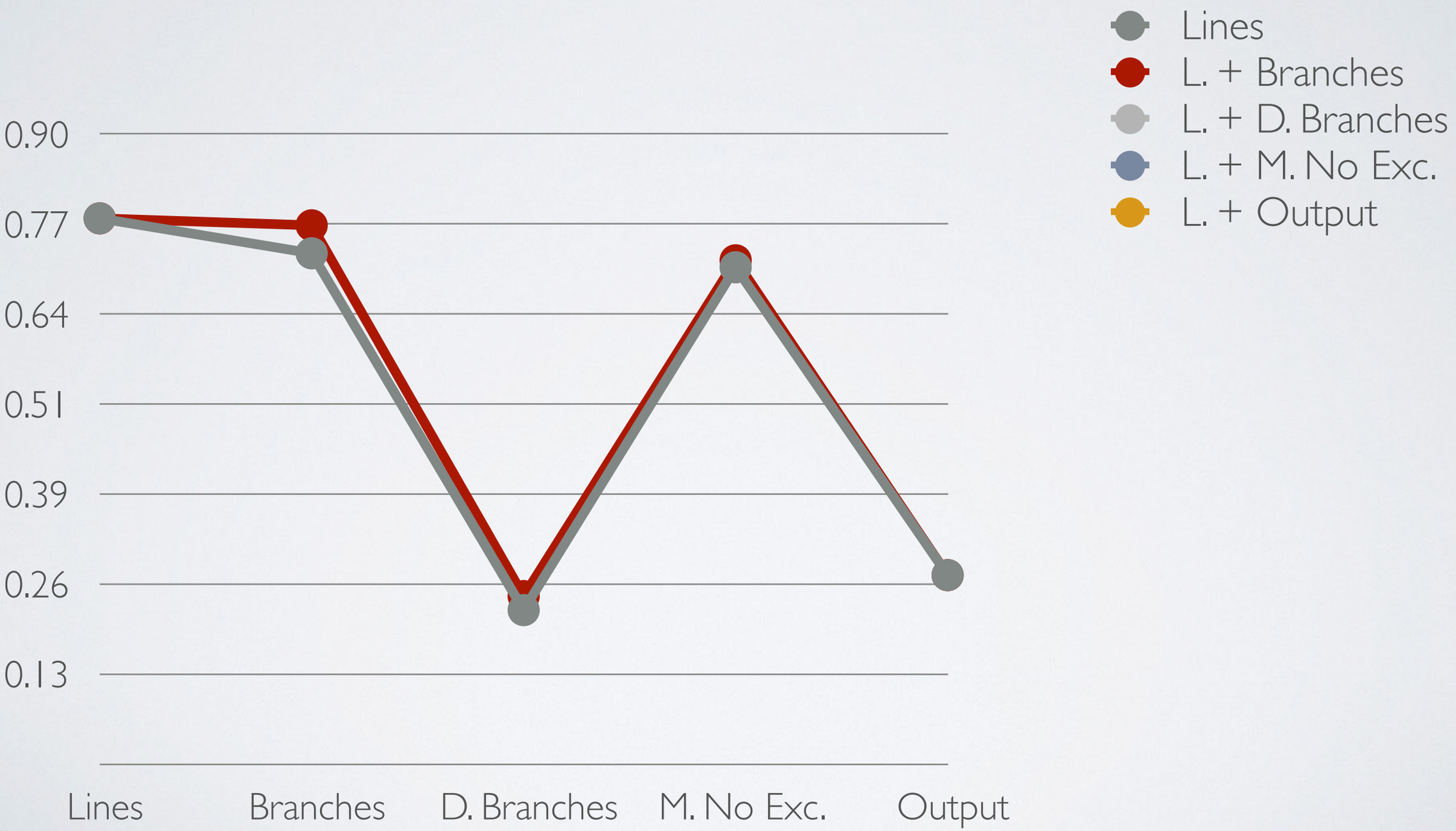
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



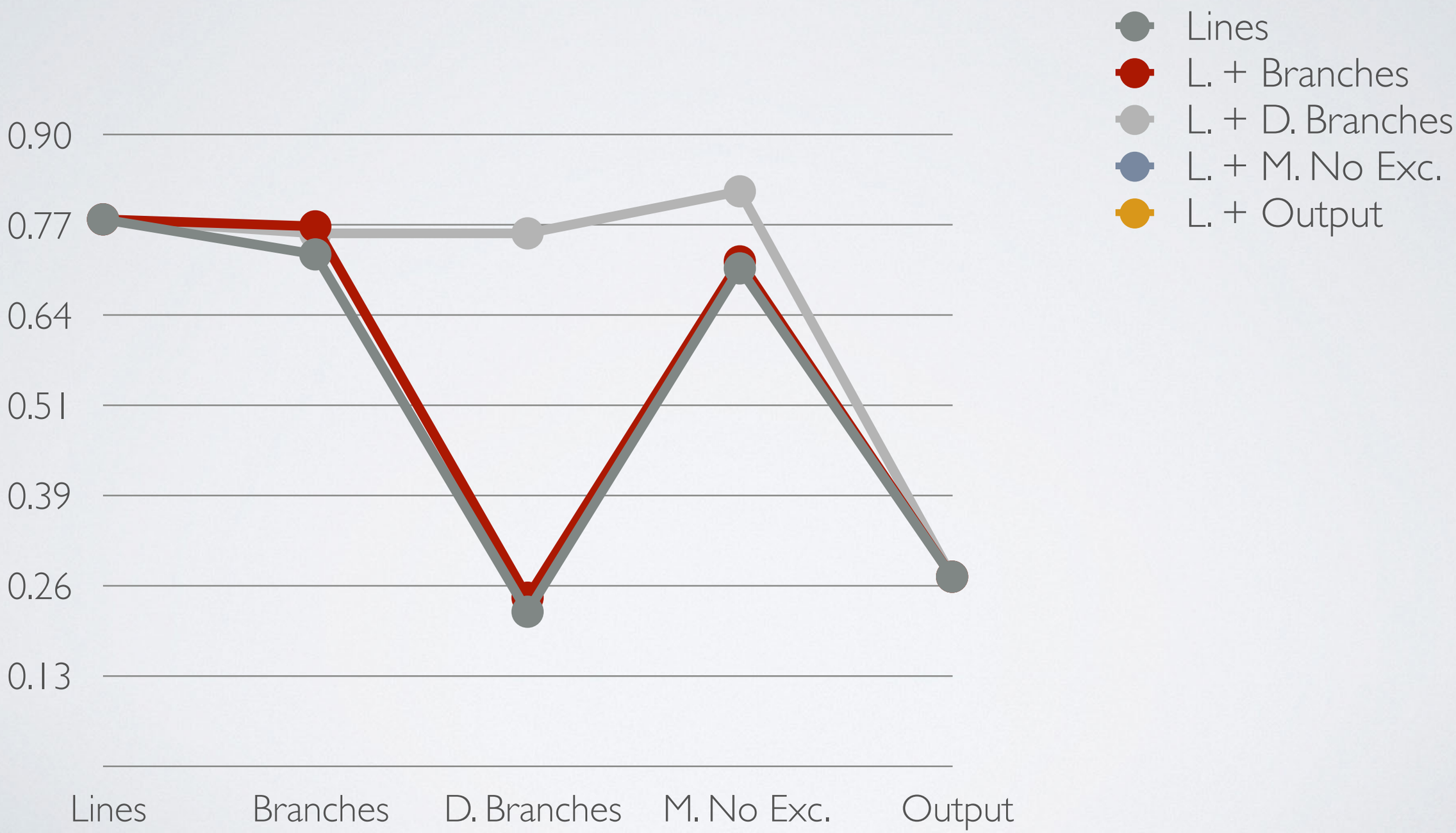
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



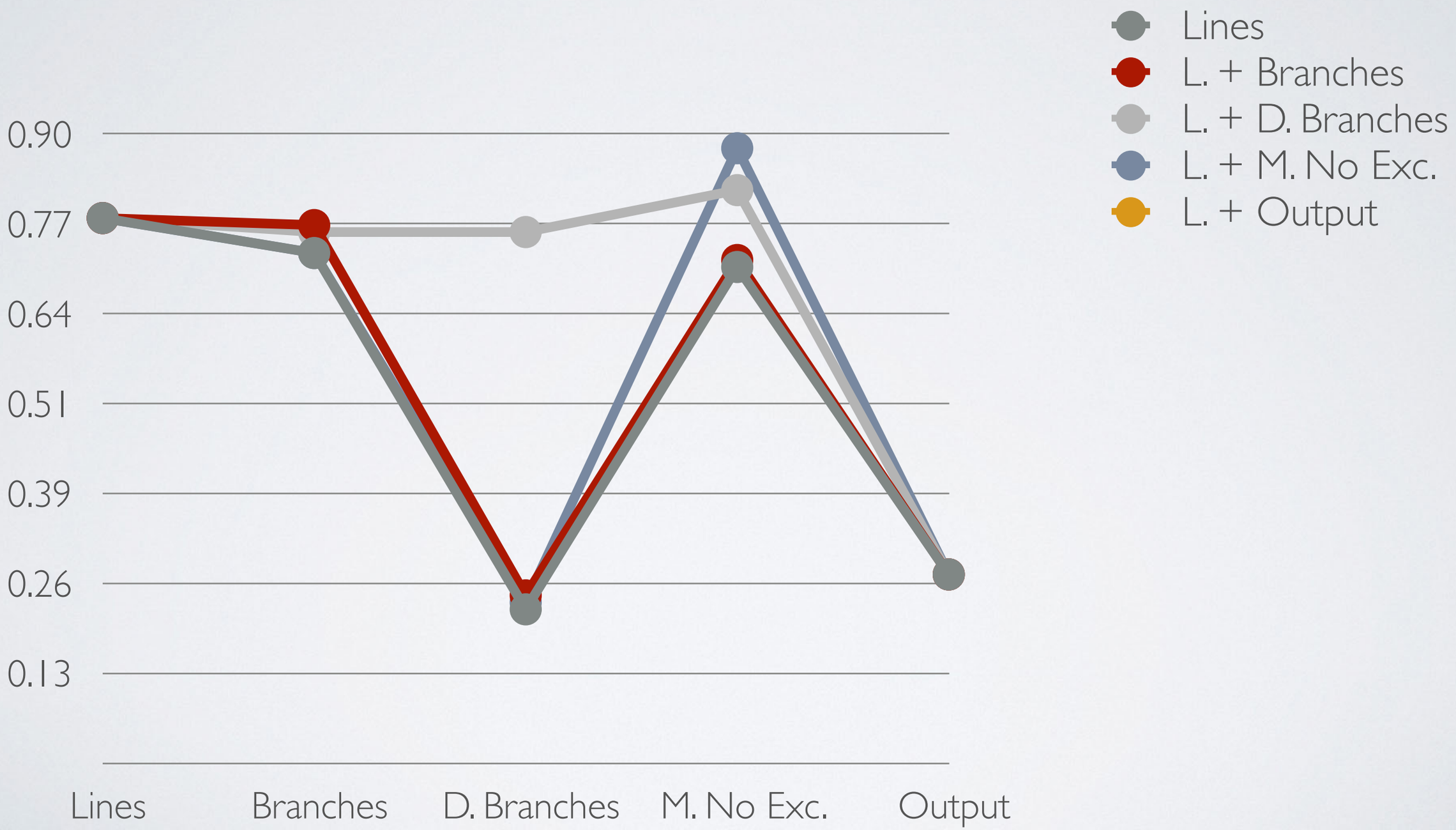
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



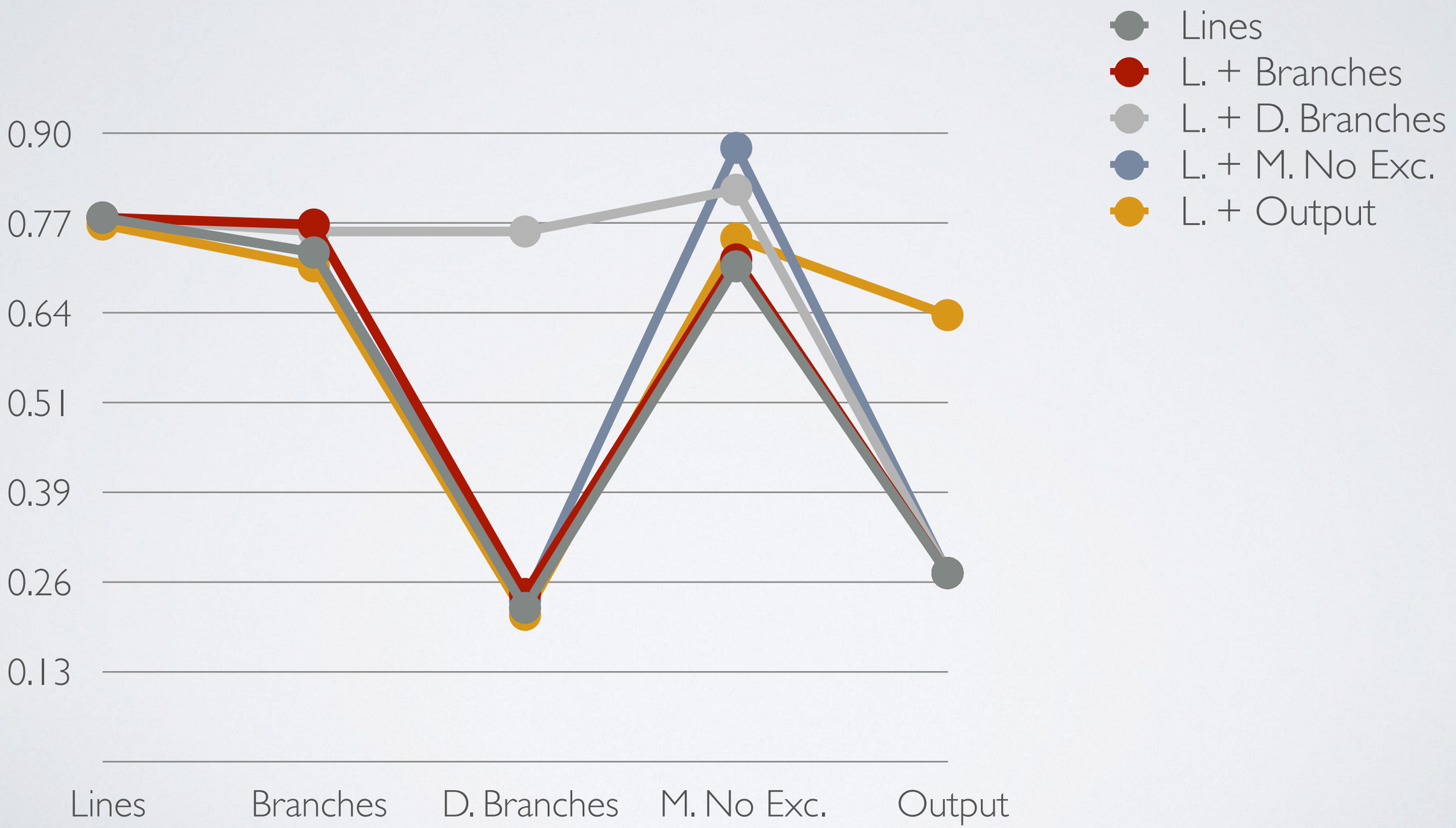
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



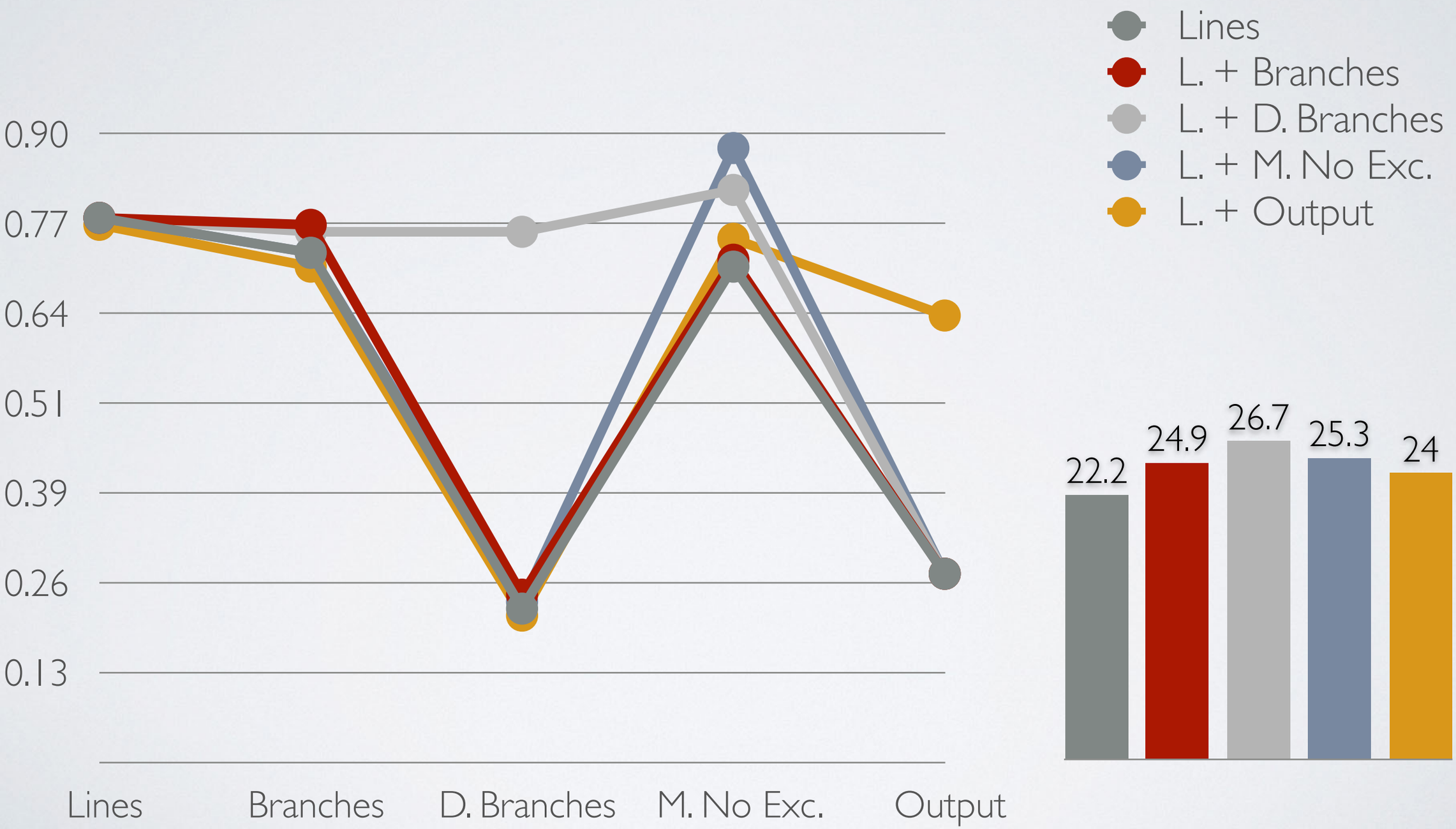
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



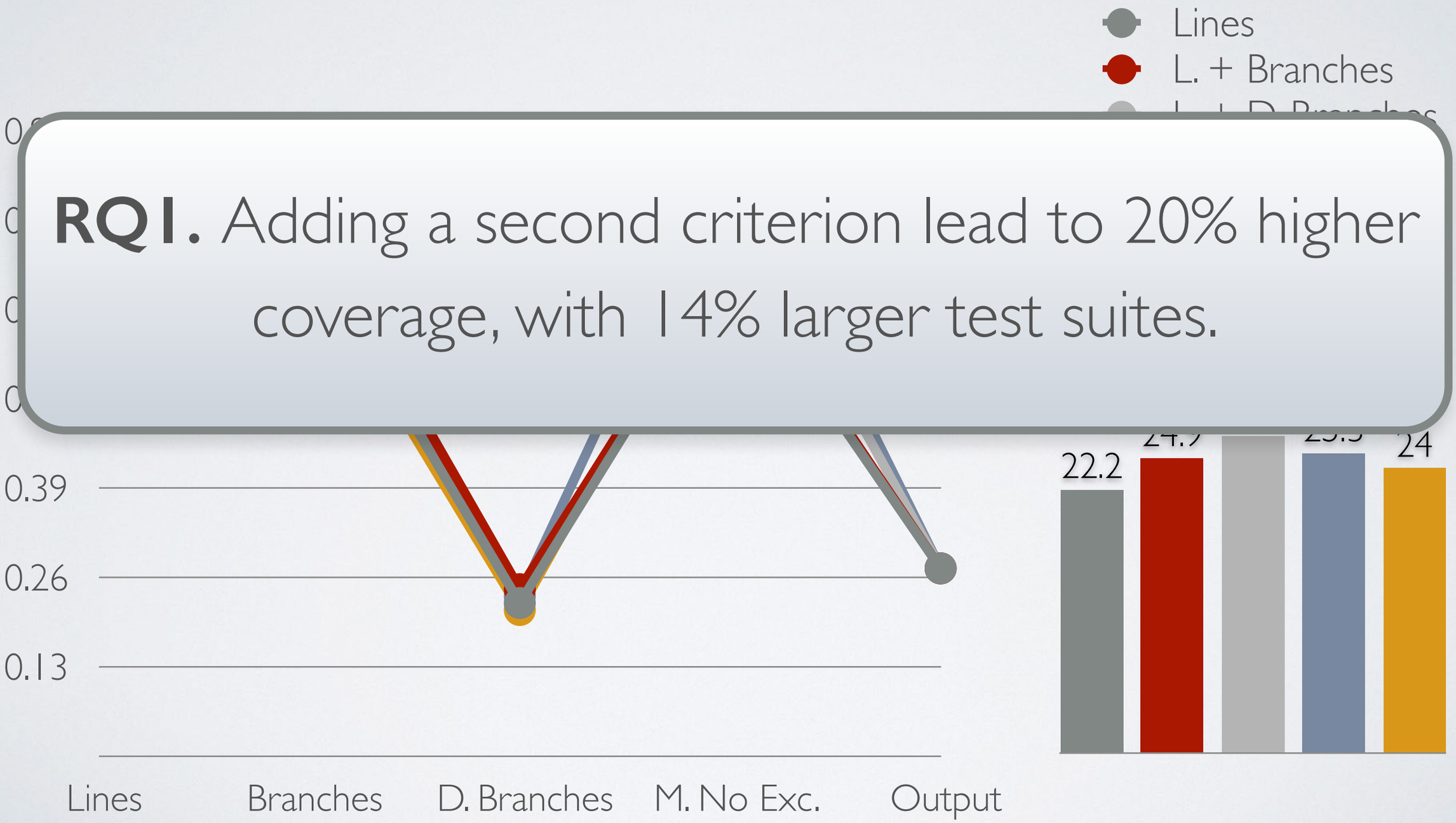
TEST SUITE COVERAGE AND SIZE

Adding a second coverage criterion



TEST SUITE COVERAGE AND SIZE

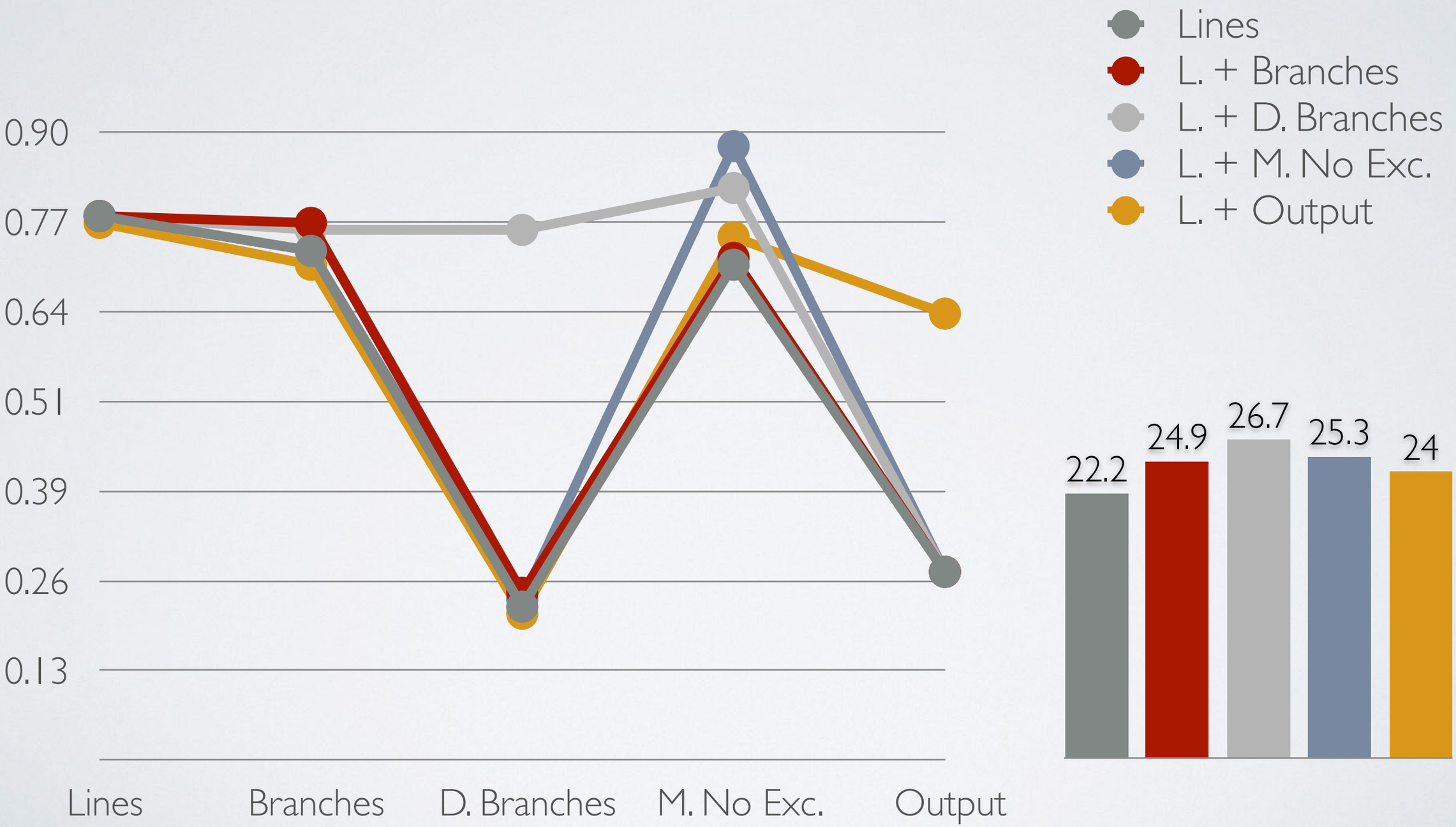
Adding a second coverage criterion



RQ1. Adding a second criterion lead to 20% higher coverage, with 14% larger test suites.

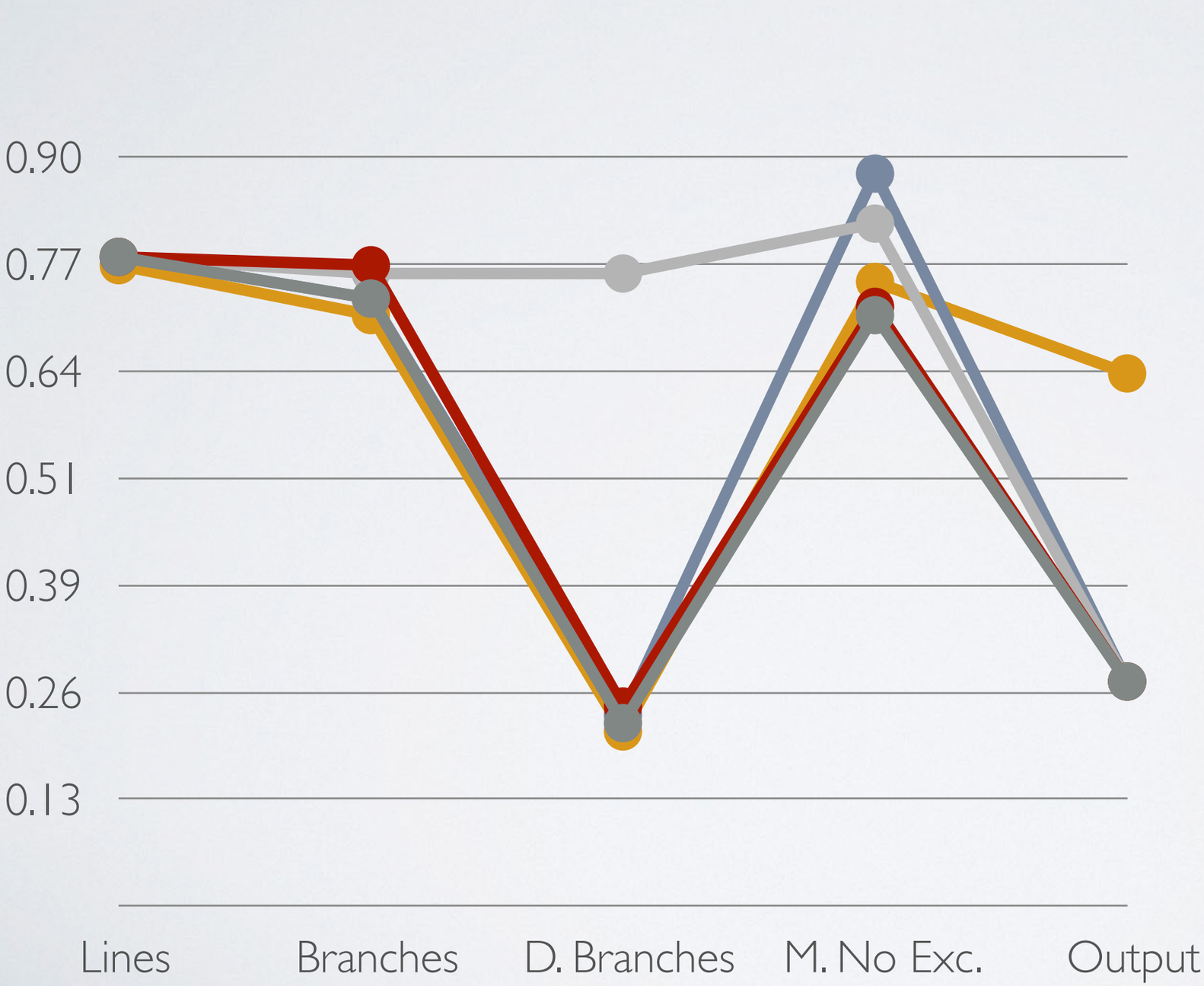
TEST SUITE COVERAGE AND SIZE

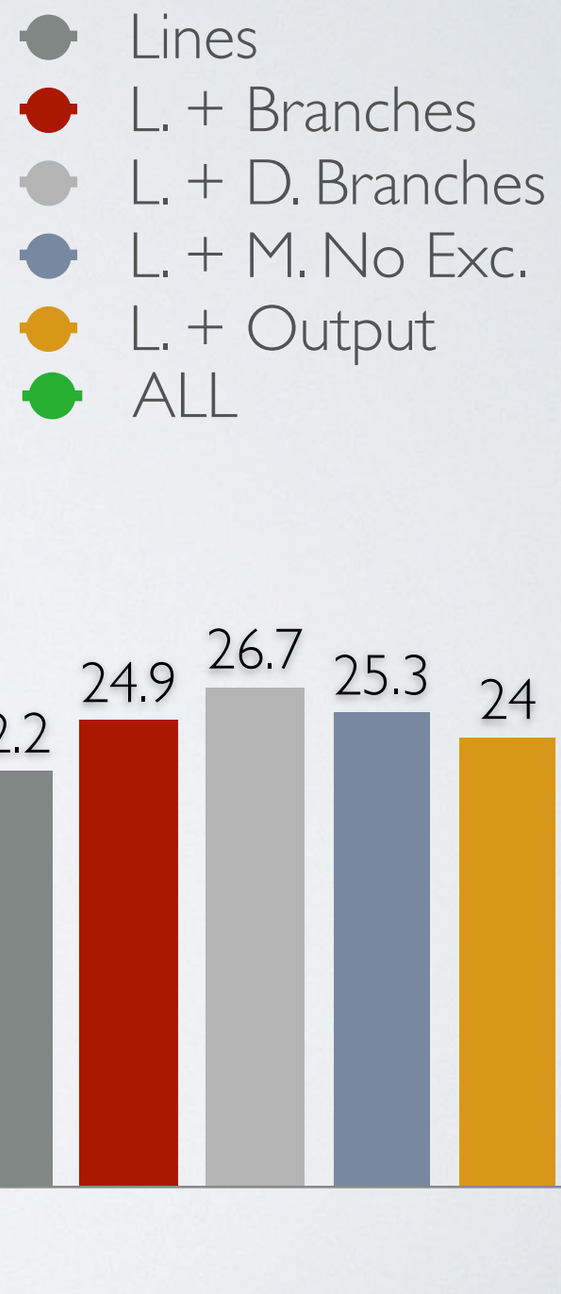
Adding a second coverage criterion



TEST SUITE COVERAGE AND SIZE

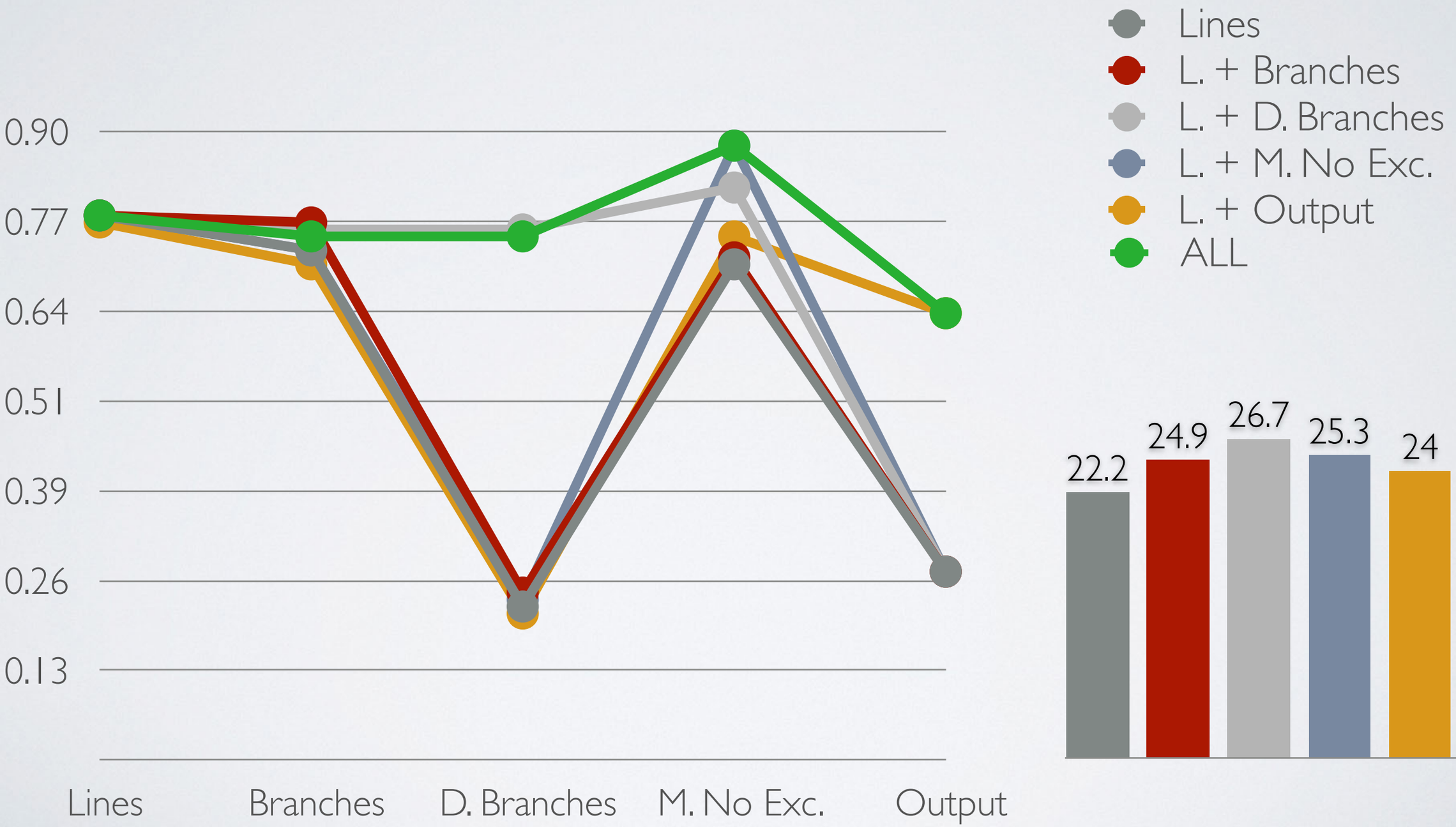
Combining **all** coverage criteria





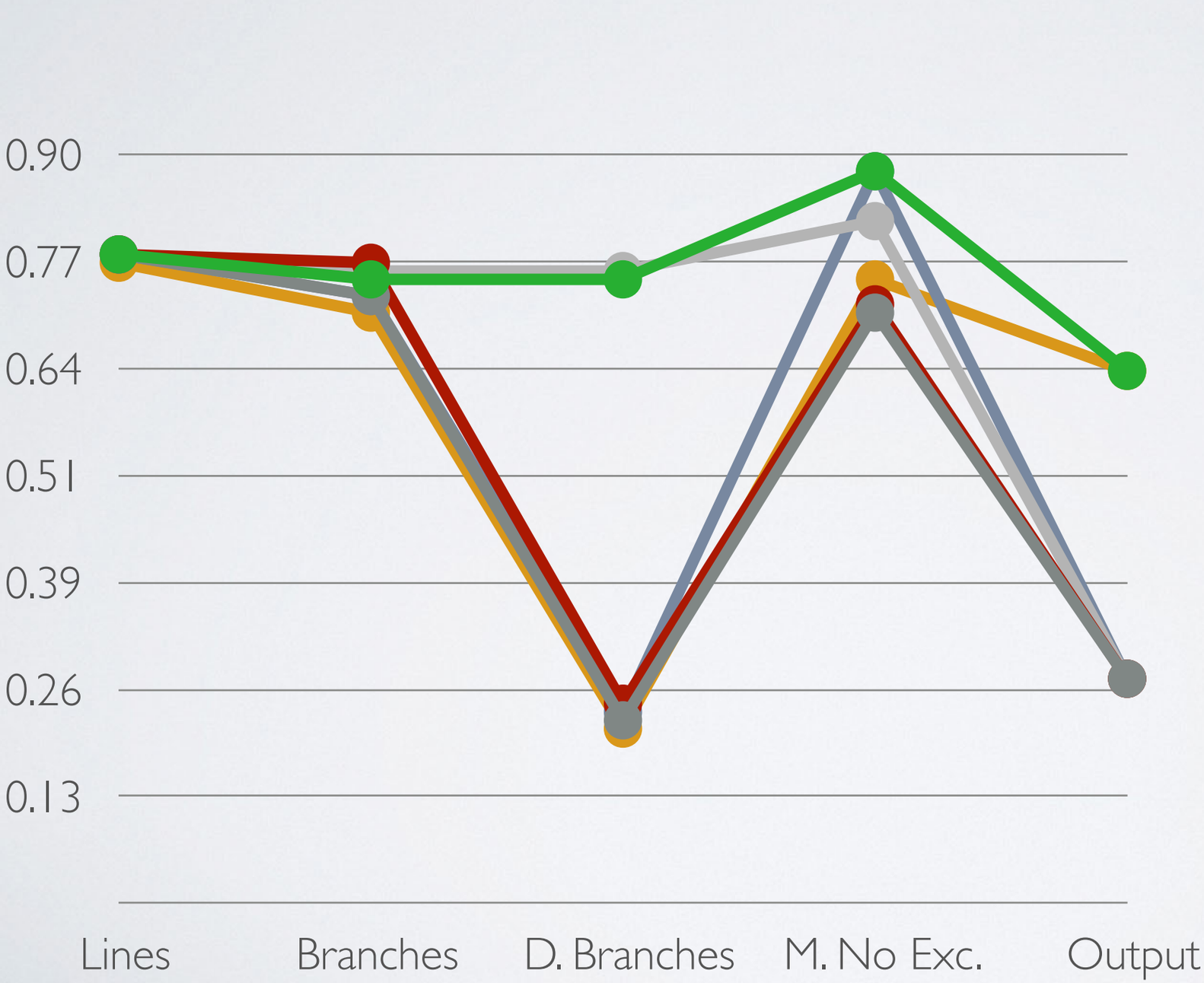
TEST SUITE COVERAGE AND SIZE

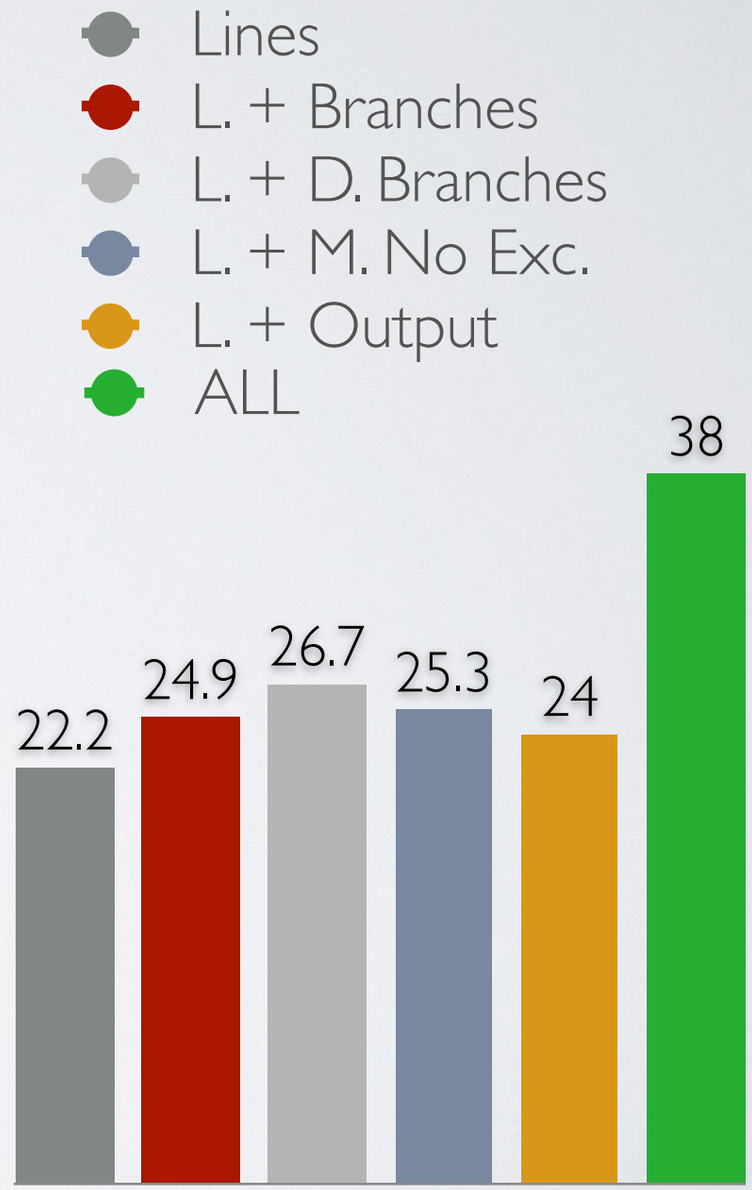
Combining **all** coverage criteria



TEST SUITE COVERAGE AND SIZE

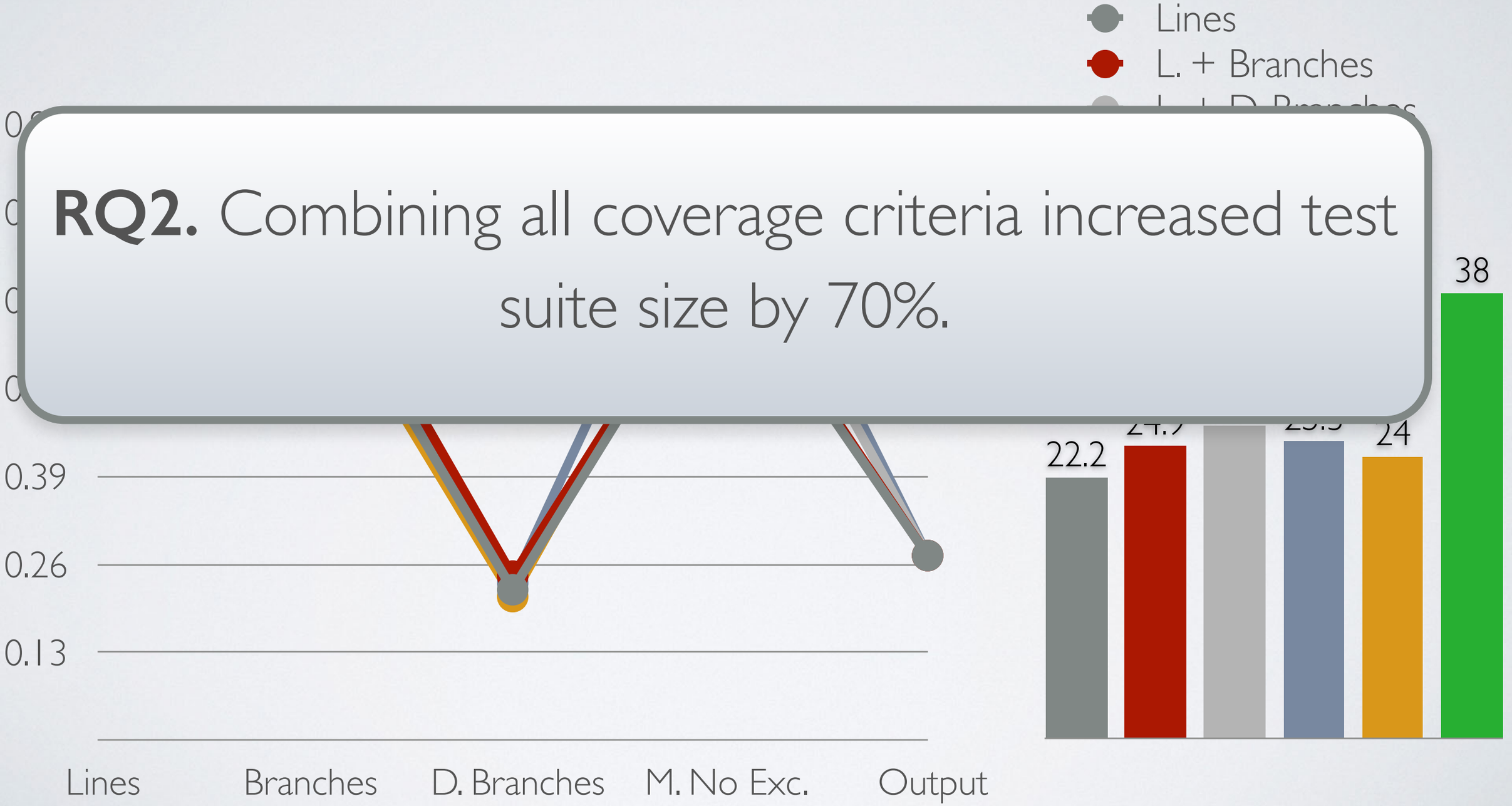
Combining **all** coverage criteria





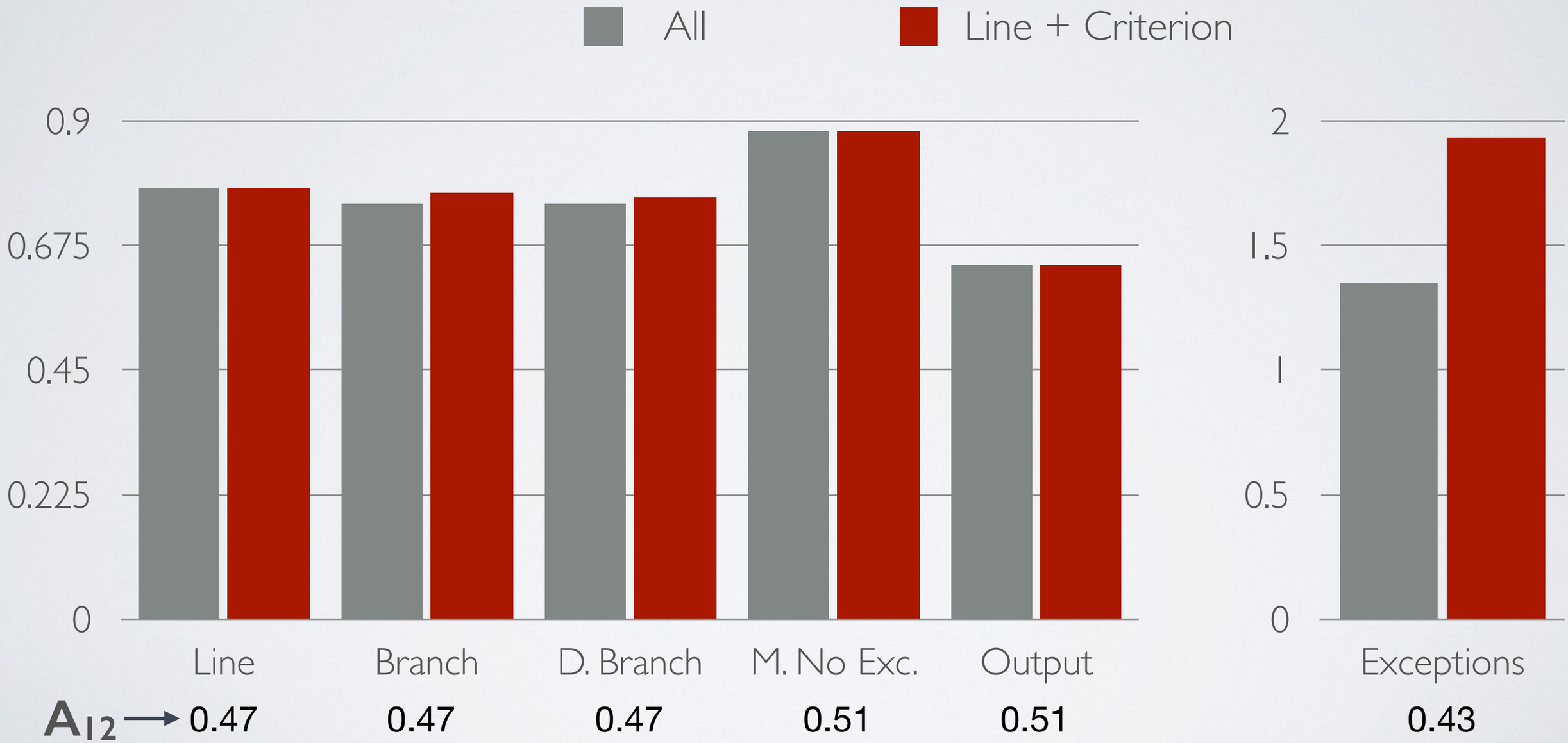
TEST SUITE COVERAGE AND SIZE

Combining **all** coverage criteria



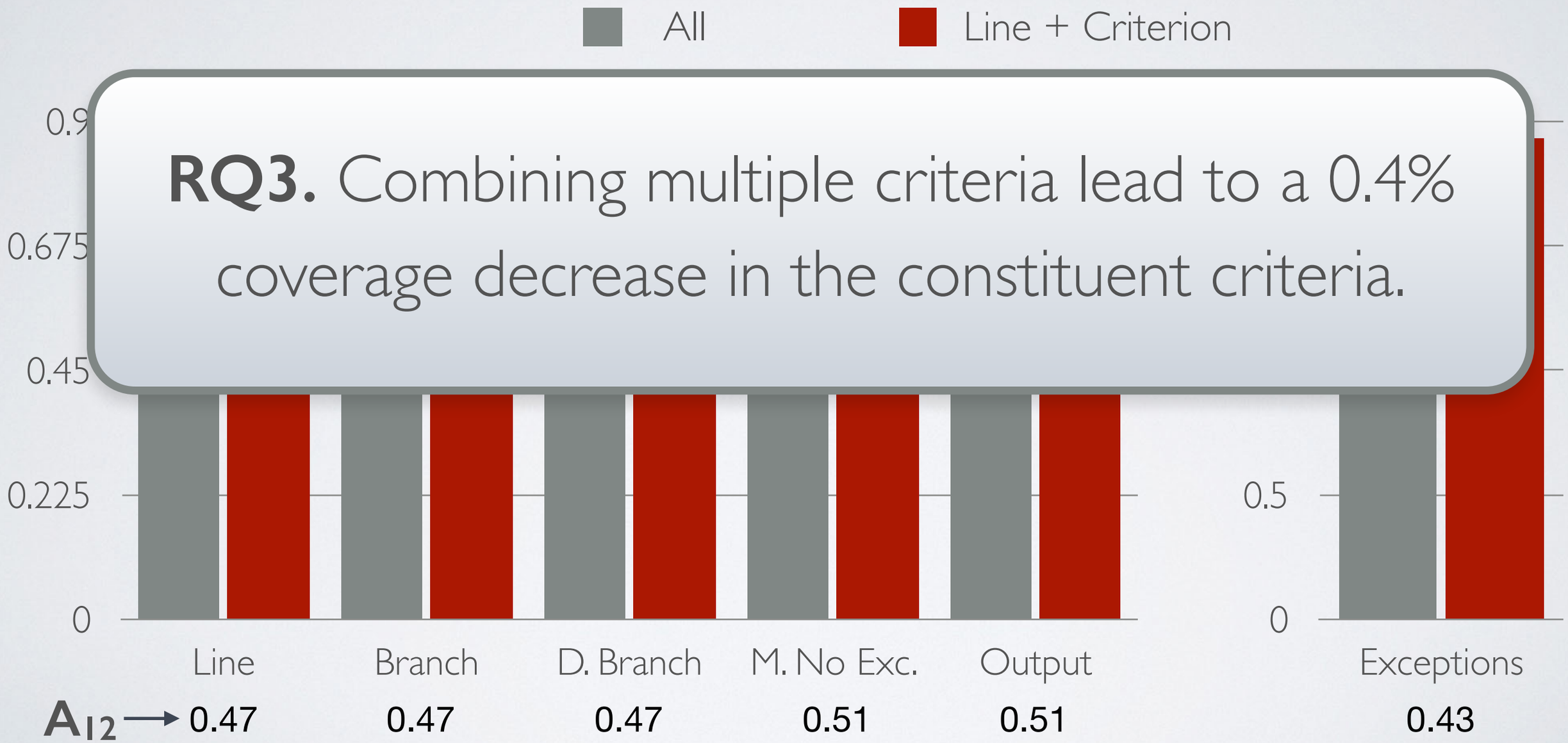
RQ2. Combining all coverage criteria increased test suite size by 70%.

COVERAGE OF CONSTITUENT CRITERIA



***p*-values <= 0.001**

COVERAGE OF CONSTITUENT CRITERIA

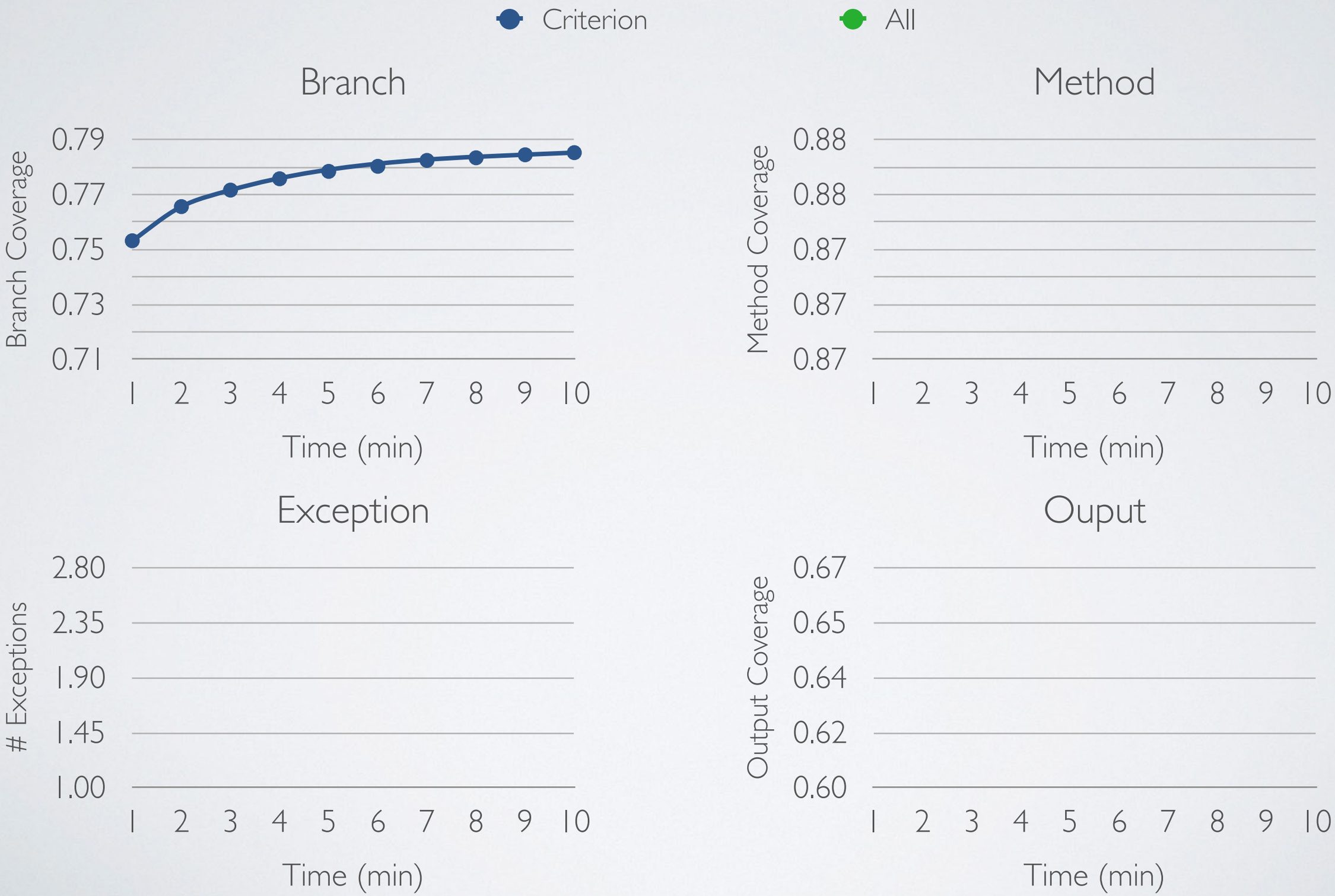


RQ3. Combining multiple criteria lead to a 0.4% coverage decrease in the constituent criteria.

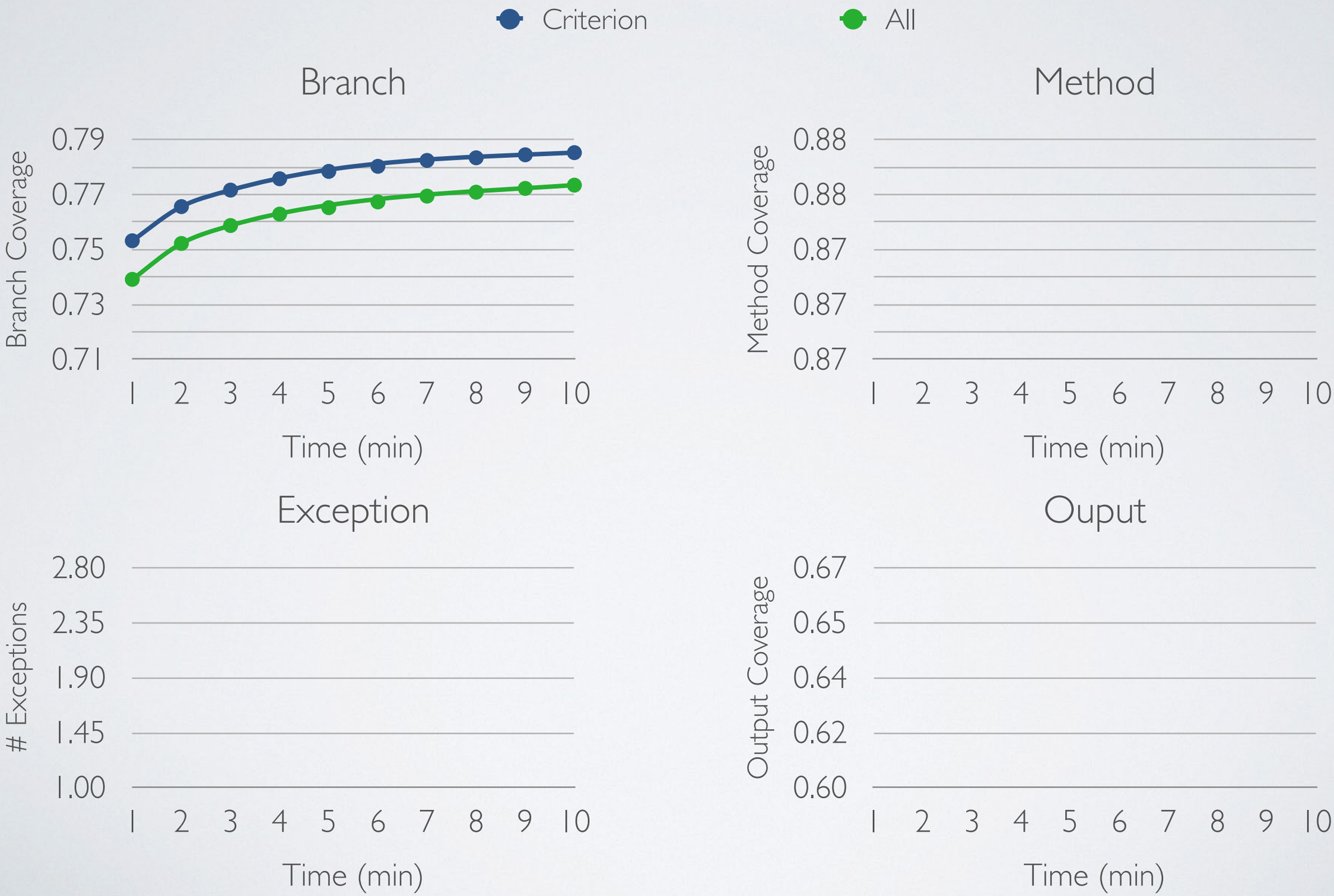
***p*-values <= 0.001**

INCREASED SEARCH BUDGET

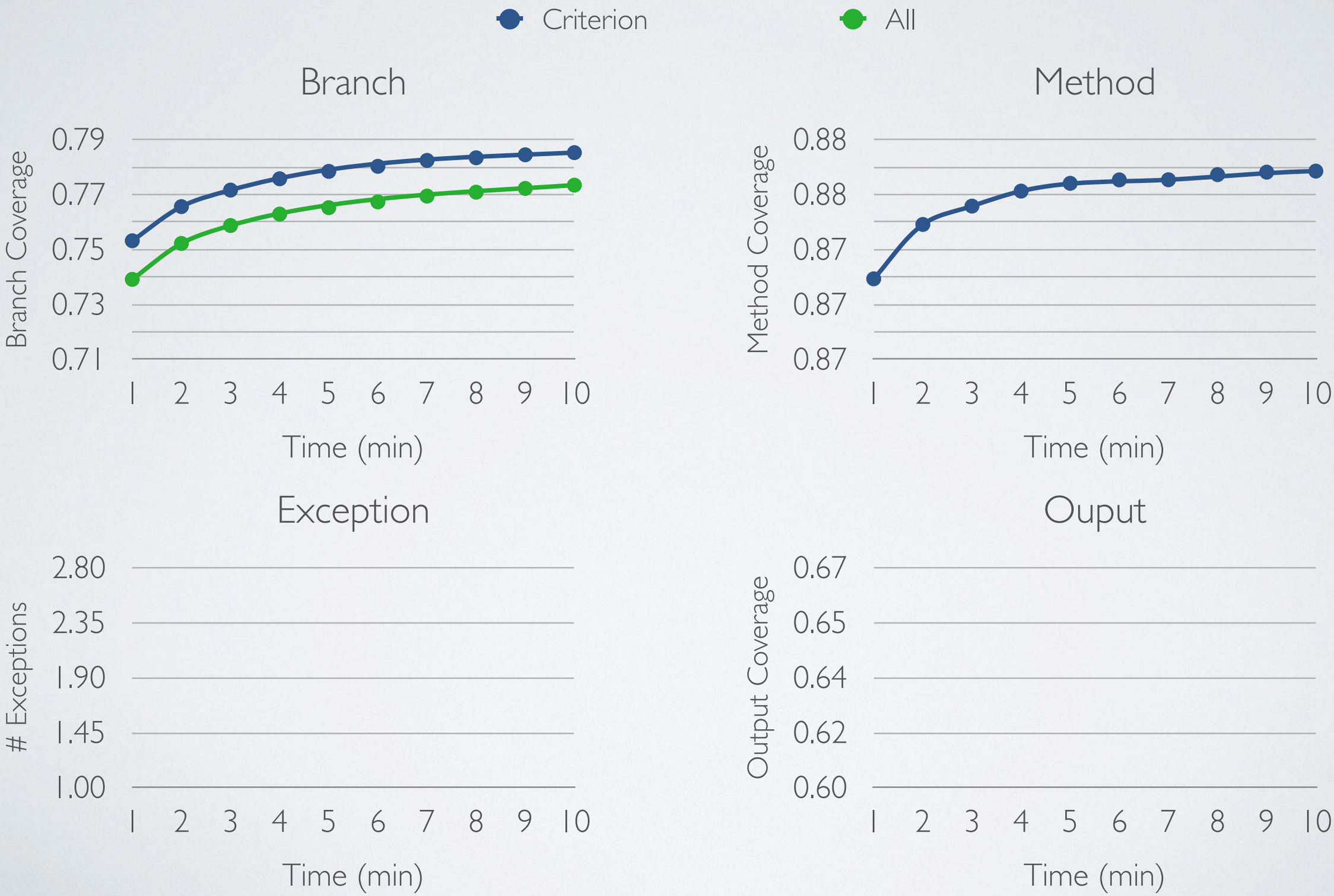
INCREASED SEARCH BUDGET



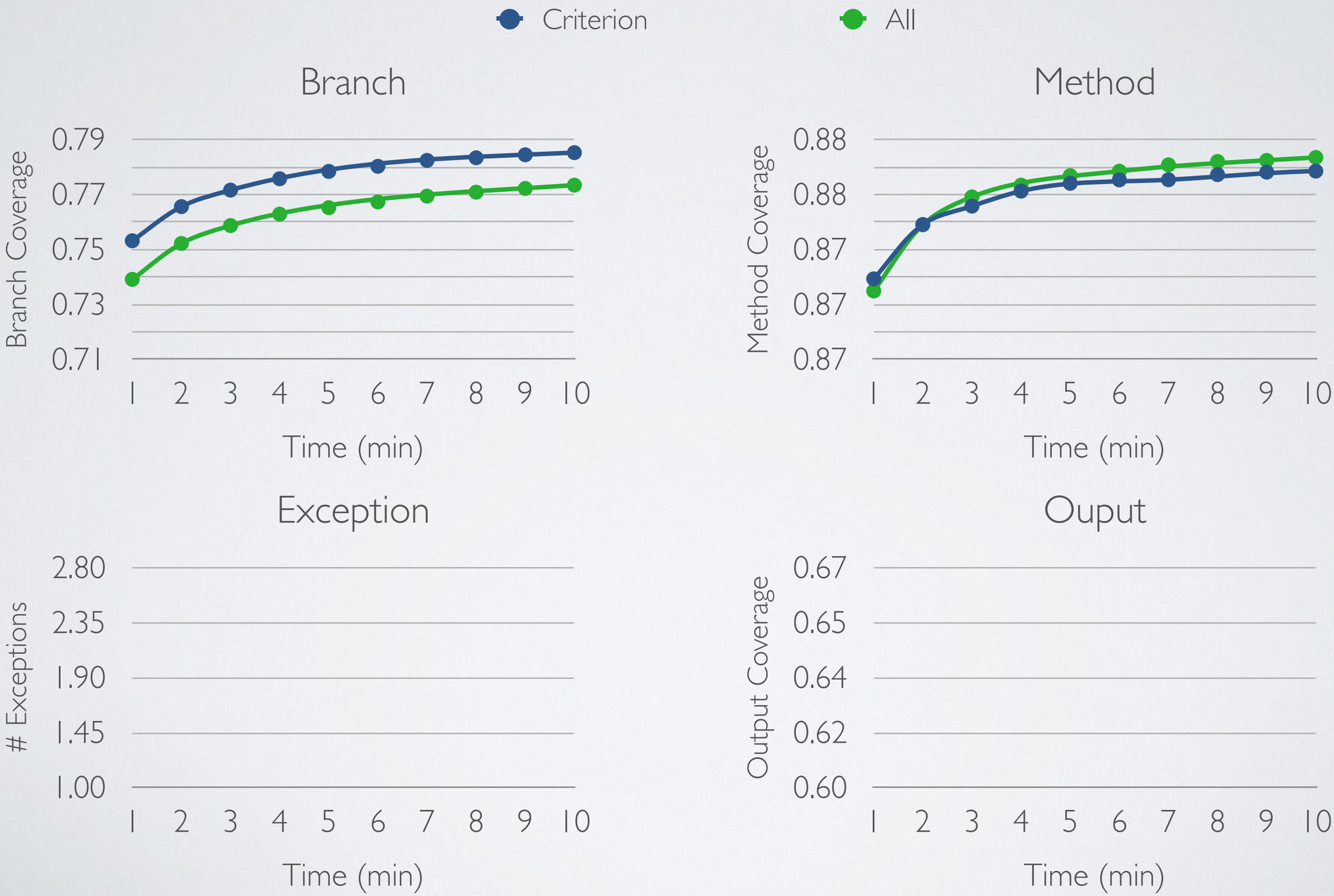
INCREASED SEARCH BUDGET



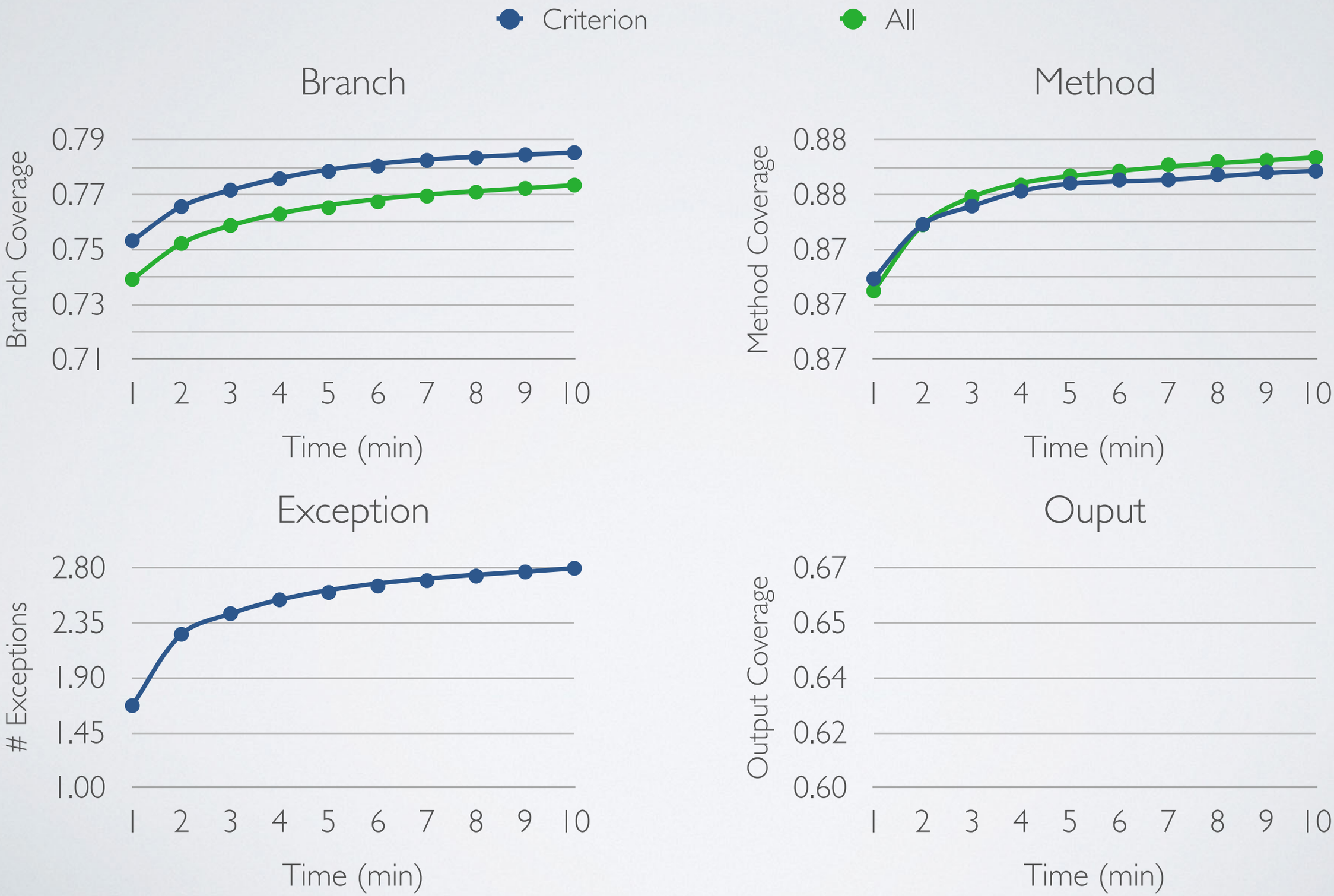
INCREASED SEARCH BUDGET



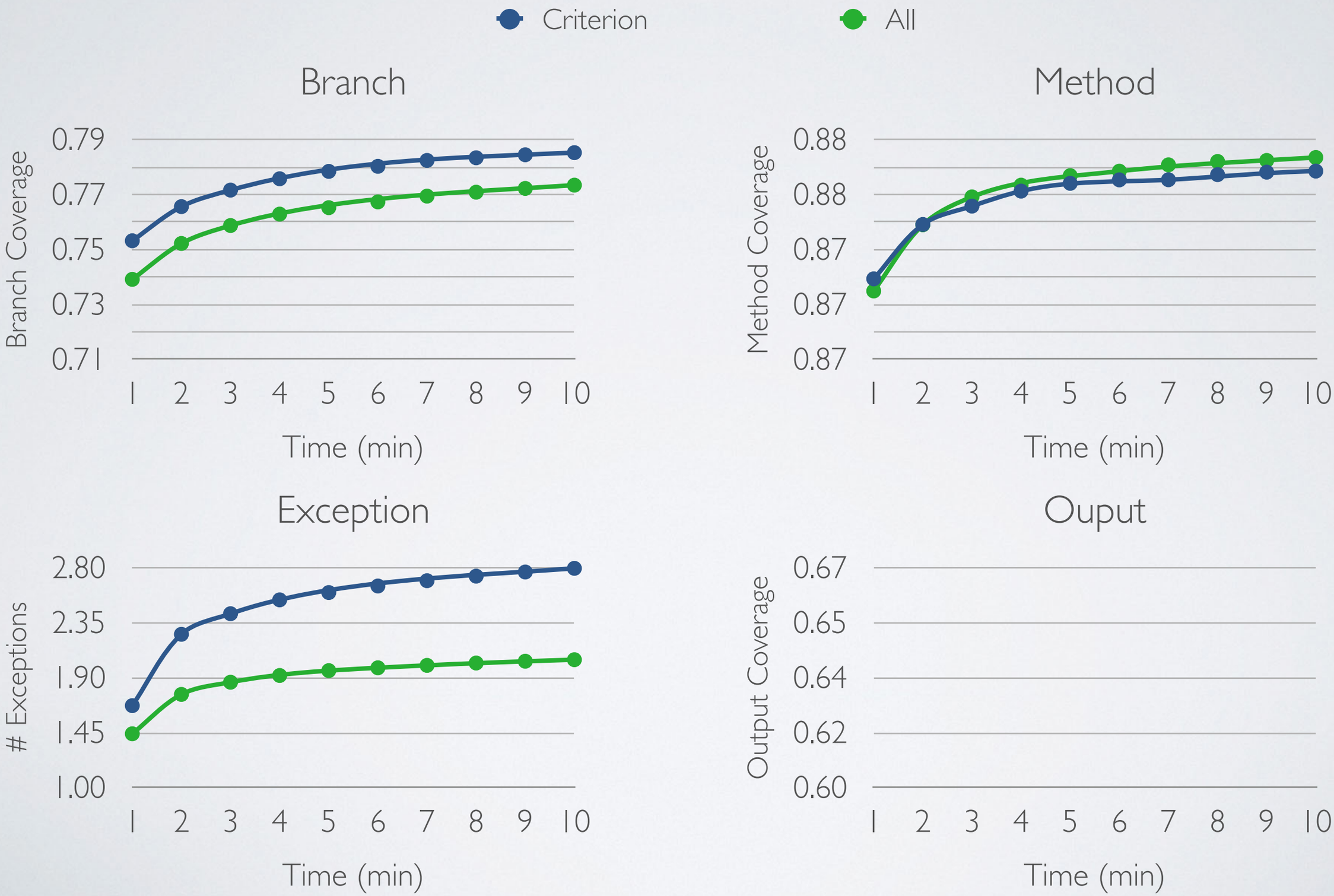
INCREASED SEARCH BUDGET



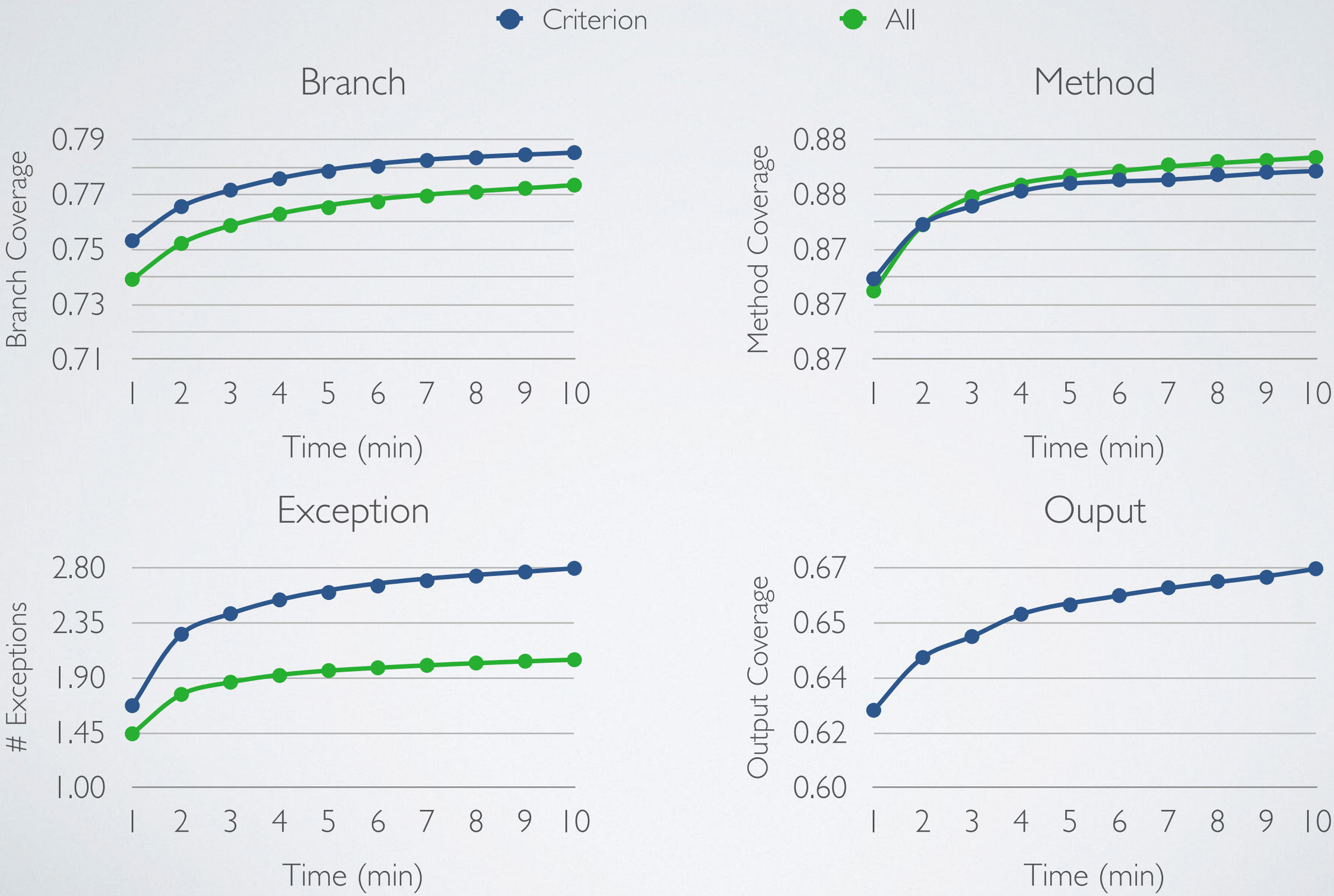
INCREASED SEARCH BUDGET



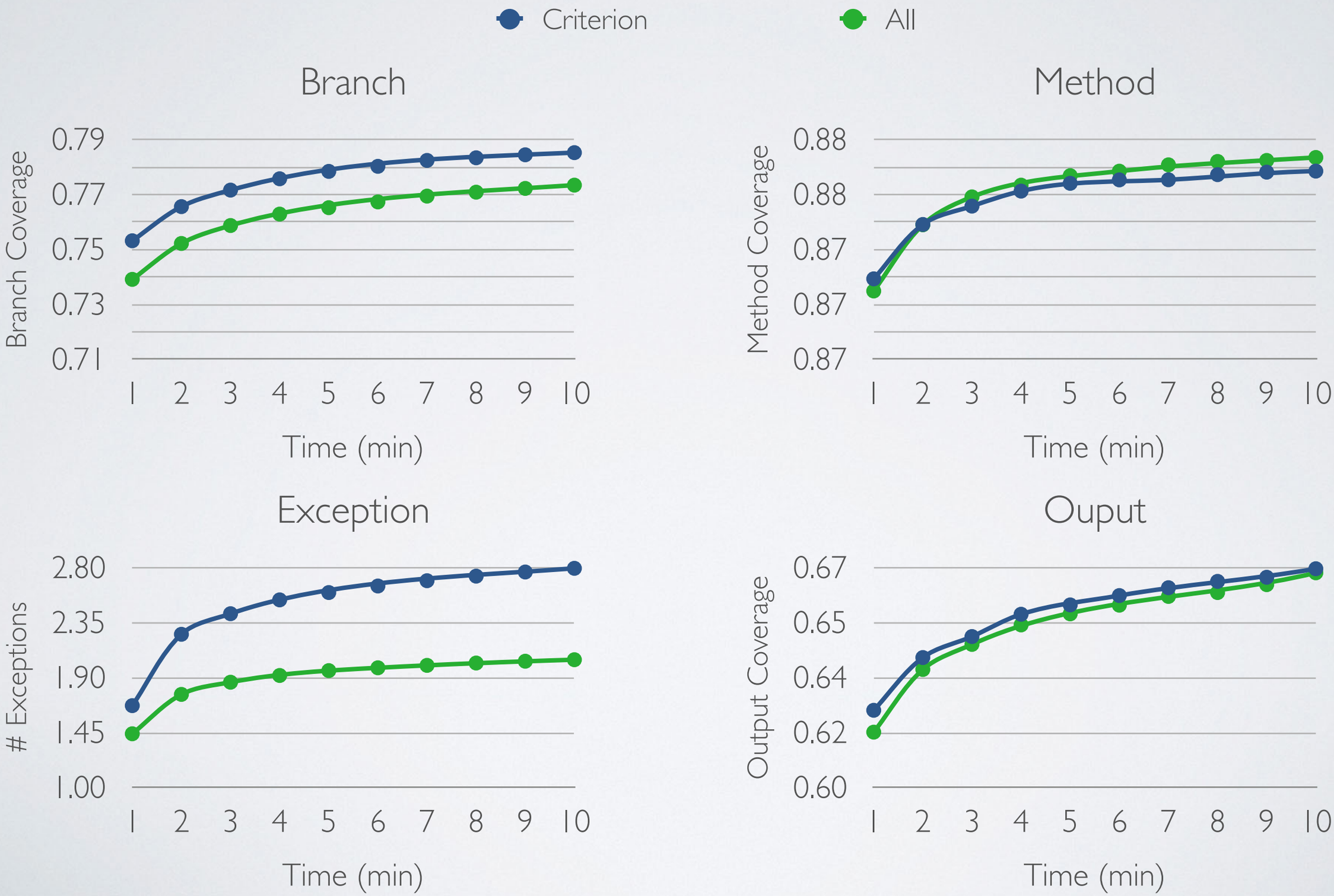
INCREASED SEARCH BUDGET



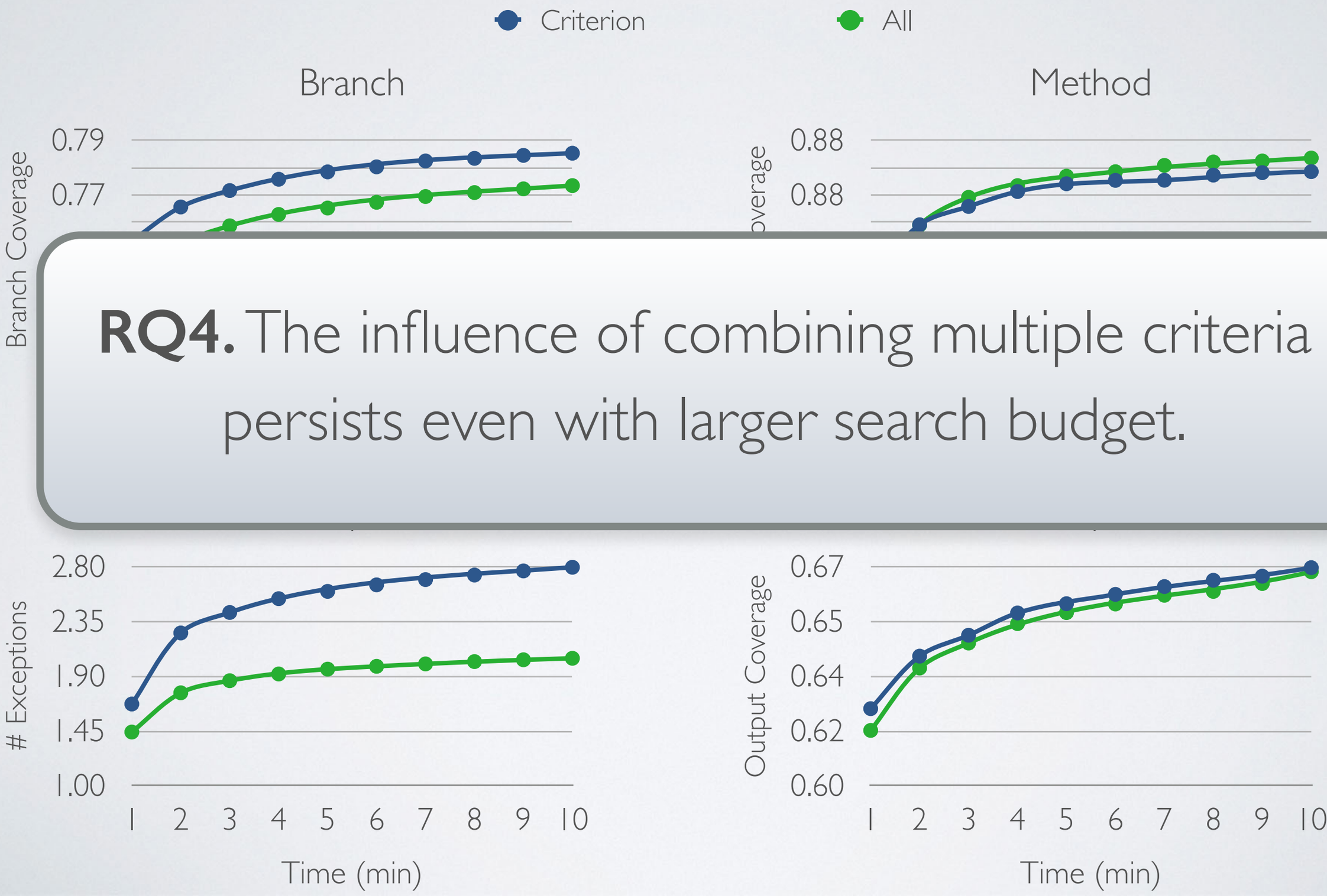
INCREASED SEARCH BUDGET



INCREASED SEARCH BUDGET



INCREASED SEARCH BUDGET



RQ4. The influence of combining multiple criteria persists even with larger search budget.

DEMO

Eclipse plugin update site

http://www.evosuite.org/update/

CONCLUSIONS

CONCLUSIONS

- The desired properties of a test suite are multi-faceted

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
- Combination of multiple criteria is feasible

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
- Combination of multiple criteria is feasible
- Really useful for practitioners?

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
  - Combination of multiple criteria is feasible
  - Really useful for practitioners?
- Non-functional criteria: readability, execution time

CONCLUSIONS

- The desired properties of a test suite are multi-faceted
- Practical coverage criteria to guide search-based test generation
  - Combination of multiple criteria is feasible
  - Really useful for practitioners?
- Non-functional criteria: readability, execution time
  - Conflicting optimisation goals