Generating Readable Unit Tests for Guava

Ermira Daka, José Campos, and Gordon Fraser

University of Sheffield

Jonathan Dorn and Westley Weimer

University of Virginia

ESEC/FSE

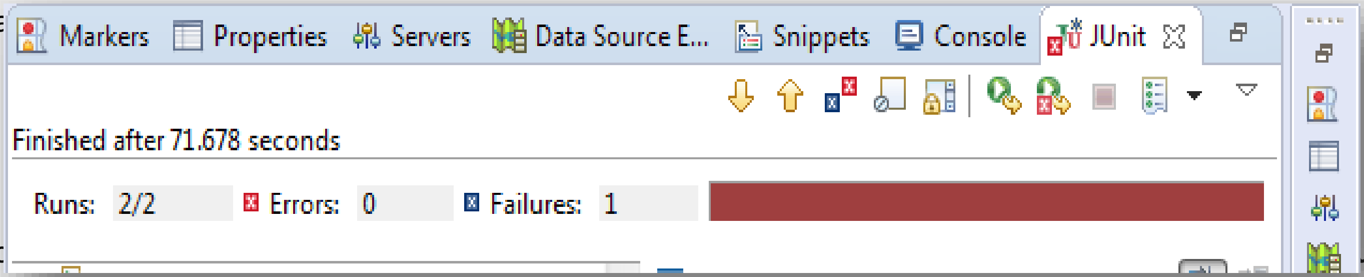
2015

“[Developers] read tests […] 77% of the total time they spend in them”

Moritz Beller, Georgios Gousios, Annibale Panichella, and Andy Zaidman. When, How, and Why Developers (Do Not) Test in Their IDEs. FSE 2015

```java
public final class MoreExecutors {
  private MoreExecutors() {}

 …
  public static Executor directExecutor() {
    return DirectExecutor.INSTANCE;
  }
…

  public static ListeningExecutorService listeningDecorator(
      ExecutorService delegate) {
    return (delegate instanceof ListeningExecutorService)
        ? (ListeningExecutorService) delegate
        : (delegate instanceof ScheduledExecutorService)
        ? new ScheduledListeningDecorator((ScheduledExecutorService) delegate)
        : new ListeningDecorator(delegate);
  }

 …
}
```

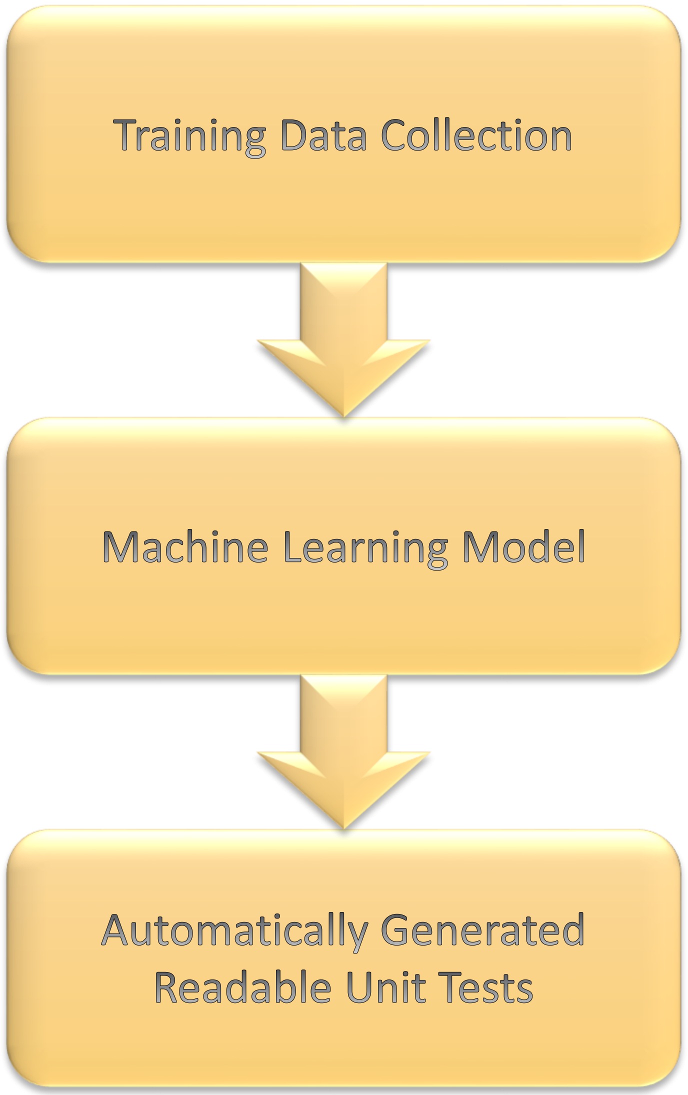
```
 public class MoreExecutors_ESTest extends MoreExecutors_ESTest_scaffolding {

   @Test
   public void test0()  throws Throwable  {
       Executor executor0 = MoreExecutors.directExecutor();
       MockThread mockThread0 = new MockThread();
       int int0 = 0;
       ScheduledThreadPoolExecutor scheduledThreadPoolExecutor0 = new ScheduledThreadPoolExecutor(int0);
       ListeningScheduledExecutorService listeningScheduledExecutorService0 =
MoreExecutors.listeningDecorator((ScheduledExecutorService) scheduledThreadPoolExecutor0);
       long long0 = 1L;
       TimeUnit timeUnit0 = TimeUnit.MILLISECONDS;
       int int1 = (-774);
       LinkedBlockingDeque<Runnable> linkedBlockingDeque0 = new LinkedBlockingDeque<Runnable>();
       ThreadPoolExecutor.DiscardOldestPolicy threadPoolExecutor_DiscardOldestPolicy0 = new ThreadPoolExecutor.DiscardOldestPolicy();
       int int2 = 0;
       int int3 = 431;
       TimeUnit timeUnit1 = TimeUnit.MINUTES;
       ThreadPoolExecutor threadPoolExecutor0 = new ThreadPoolExecutor(int0, int3, long0, timeUnit1, (BlockingQueue<Runnable>)
linkedBlockingDeque0, (RejectedExecutionHandler) threadPoolExecutor_DiscardOldestPolicy0);
       BlockingQueue<Runnable> blockingQueue0 = threadPoolExecutor0.getQueue();
       ThreadFactory threa
       TimeUnit timeUnit2
       int int4 = (-188);
       TimeUnit timeUnit3
       ThreadPoolExecutor
       try {
         threadPoolExecutc                                                                    ue<Runnable>)
linkedBlockingDeque0, threadFactory0);
       } catch(IllegalArgumentException e) {
          //
          // no message in exception (getMessage() returned null)
          //
       }
   }
 }
```



Finished after 71.678 seconds

Runs: 2/2 Errors: 0 Failures: 1

Test Readability Model



Modeling Readability to Improve Unit Tests

Ermira Daka, José Campos, and Gordon Fraser
University of Sheffield
Sheffield, UK

Jonathan Dorn and Westley Weimer
University of Virginia
Virginia, USA

ABSTRACT

Writing good unit tests can be tedious and error prone, but even once they are written, the job is not done: Developers need to reason about unit tests throughout software development and evolution, in order to diagnose test failures, maintain the tests, and to understand code written by other developers. Unreadable tests are more difficult to maintain and lose some of their value to developers. To overcome this problem, we propose a domain-specific model of unit test readability based on human judgements, and use this model to augment automated unit test generation. The resulting approach can automatically generate test suites with both high coverage and also improved readability. In human studies users prefer our improved tests and are able to answer maintenance questions about them 14% more quickly at the same level of accuracy.

Categories and Subject Descriptors. D.2.5 [Software Engineering]: Testing and Debugging – *Testing Tools*;

Keywords. Readability, unit testing, automated test generation

1. INTRODUCTION

Unit testing is a popular technique in object oriented programming, where efficient automation frameworks such as JUnit allow unit tests to be defined and executed conveniently. However, producing good tests is a tedious and error prone task, and over their lifetime, these tests often need to be read and understood by different people. Developers use their own tests to guide their implementation activities, receive tests from automated unit test generation tools to improve their test suites, and rely on the tests written by developers of other code. Any test failures require fixing either the software or the failing test, and any passing test may be consulted by developers as documentation and usage example for the code under test. Test comprehension is a manual activity that requires one to understand the behavior represented by a test — a task that may not be easy if the test was written a week ago, difficult if it was written by a different person, and challenging if the test was generated automatically.

How difficult it is to understand a unit test depends on many factors. Unit tests for object-oriented languages typically consist of sequences of calls to instantiate various objects, bring them to appropriate states, and create interactions between them. The particular choice of sequence of calls and values can have a large impact on the resulting test. For example, consider the pair of unit tests shown in Figure 1. Both tests exercise the same functionality with respect to the constructor of the class `ObjectHandlerAdaptor` in the Xineo open source project (which treats `null` and `rootHandler0` arguments identically). Despite this identical coverage of the subject class in practice, they are quite different in presentation.

```
ElementName elementName0 = new ElementName("", "");
Class<Object> class0 = Object.class;
VirtualHandler virtualHandler0 = new VirtualHandler(
    elementName0, (Class) class0);
Object object0 = new Object();
RootHandler rootHandler0 = new RootHandler((ObjectHandler
    ) virtualHandler0, object0);
ObjectHandlerAdapter objectHandlerAdapter0 = new
    ObjectHandlerAdapter((ObjectHandlerInterface)
    rootHandler0);
assertEquals("ObjectHandlerAdapter",
    objectHandlerAdapter0.getName());
```

```
ObjectHandlerAdapter objectHandlerAdapter0 = new
    ObjectHandlerAdapter((ObjectHandlerInterface) null);
assertEquals("ObjectHandlerAdapter",
    objectHandlerAdapter0.getName());
```

Figure 1: Two versions of a test that exercise the same functionality but have a different appearance and readability.

In terms of concrete features that may affect comprehension, the first test is longer, uses more different classes, defines more variables, has more parentheses, and has longer lines. The visual appearance of code in general is referred to as its *readability* — if code is not readable, intuitively it will be more difficult to perform any tasks that require understanding it. Despite significant interest from managers and developers [8], a general understanding of software readability remains elusive. For source code, Buse and Weimer [7] applied machine learning on a dataset of code snippets with human annotated ratings of readability, allowing them to predict whether code snippets are considered readable or not. Although unit tests are also just code in principle, they use a much more restricted set of language features; for example, unit tests usually do not contain conditional or looping statements. Therefore, a general code readability metric may not be well suited for unit tests.

In this paper, we address this problem by designing a domain-specific model of readability based on human judgements that applies to object oriented unit test cases. To support developers in deriving readable unit tests, we use this model in an automated approach to improve the readability of unit tests, and integrate this into an automated unit test generation tool. We present:

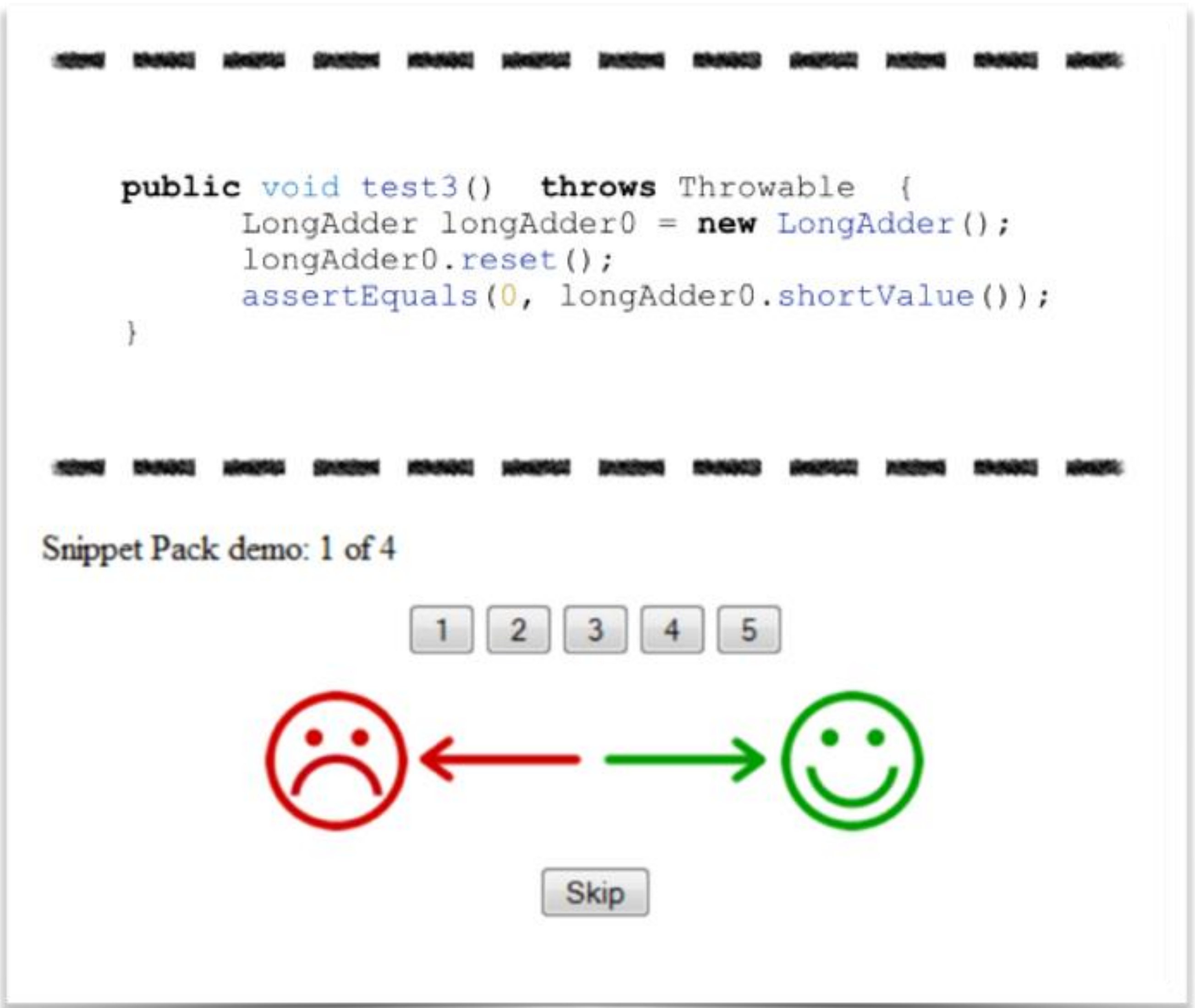
- An analysis of the syntactic features of unit tests and their importance based on human judgement (Section 2.).
- A regression model based on an optimized set of features to predict the readability of unit tests (Section 2.).

ESEC/FSE '15, August 31 - September 04, 2015, Bergamo, Italy
Copyright 2015 ACM ISBN 978-1-4503-3675-8/15/08
DOI: http://dx.doi.org/10.1145/2786805.2786838 ...$15.00

ESEC FSE 2015

Test readability judgments

amazon mechanical turk

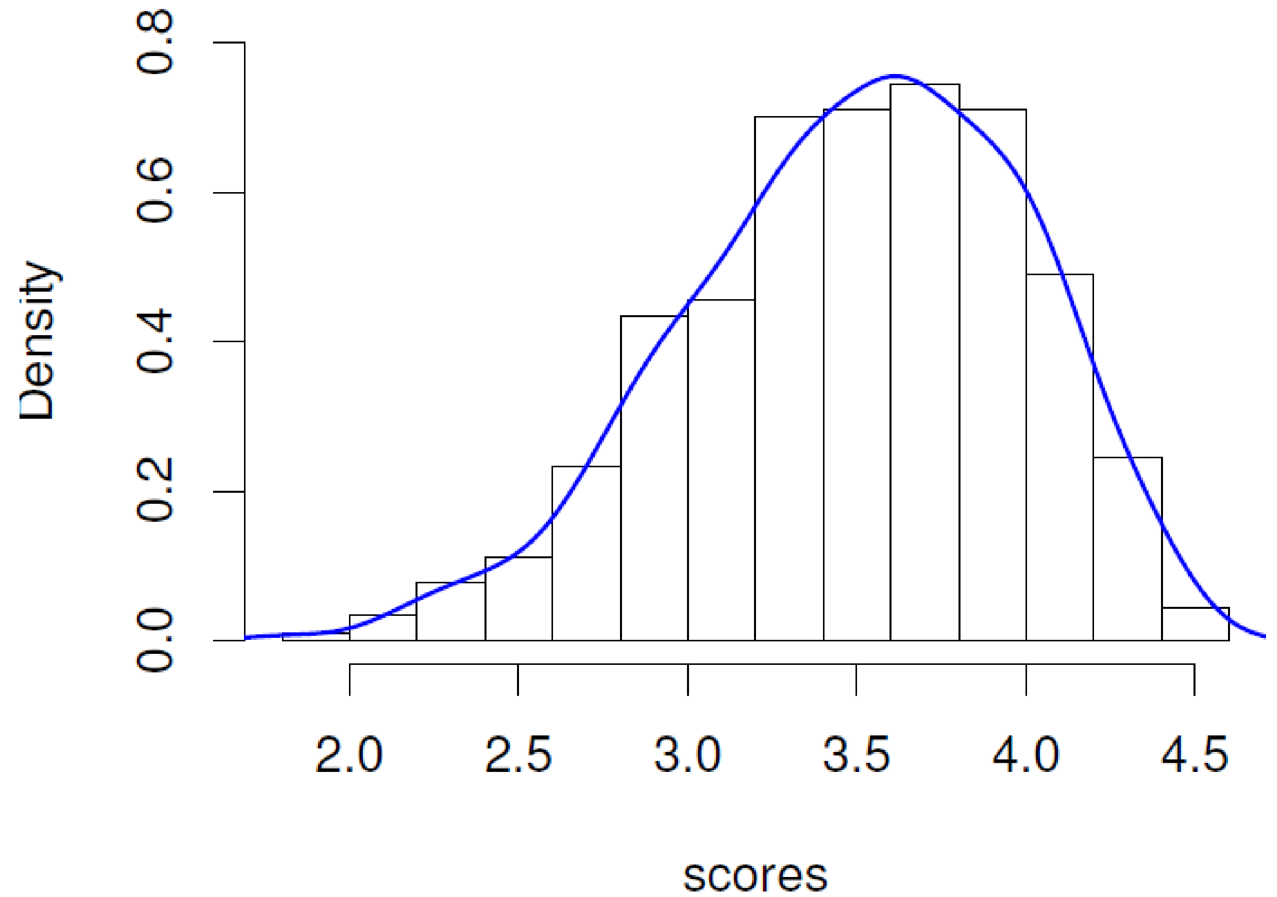


```
public void test3()  throws Throwable  {
        LongAdder longAdder0 = new LongAdder();
        longAdder0.reset();
        assertEquals(0, longAdder0.shortValue());
}
```

Snippet Pack demo: 1 of 4

Test readability judgments

- 15,669 human judgments of readability



```
public void test0()  throws Throwable  {
        String string0 = "";
        String string1 = "]";
        MessageDigestHashFunction messageDigestHashFunction0 = null;
        try {
                messageDigestHashFunction0 = new MessageDigestHashFunction(string0, string1);
        } catch(AssertionError e) {
                //
                // java.security.NoSuchAlgorithmException:  MessageDigest not available
                //
        }
}
```

Test Length = 10

Unique Identifiers = 3

Max Line Length = 77

Has Exception = True

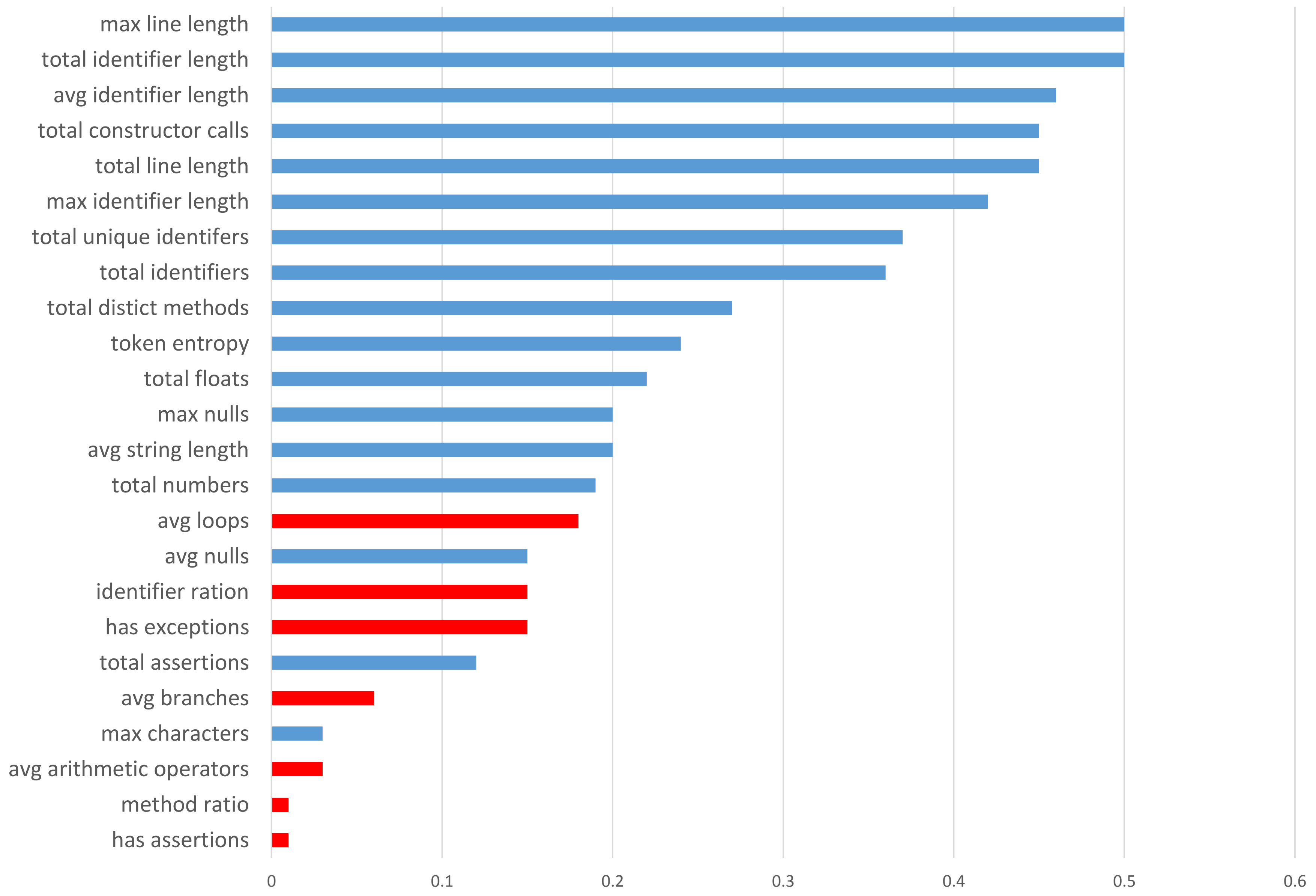
Constructors = 1

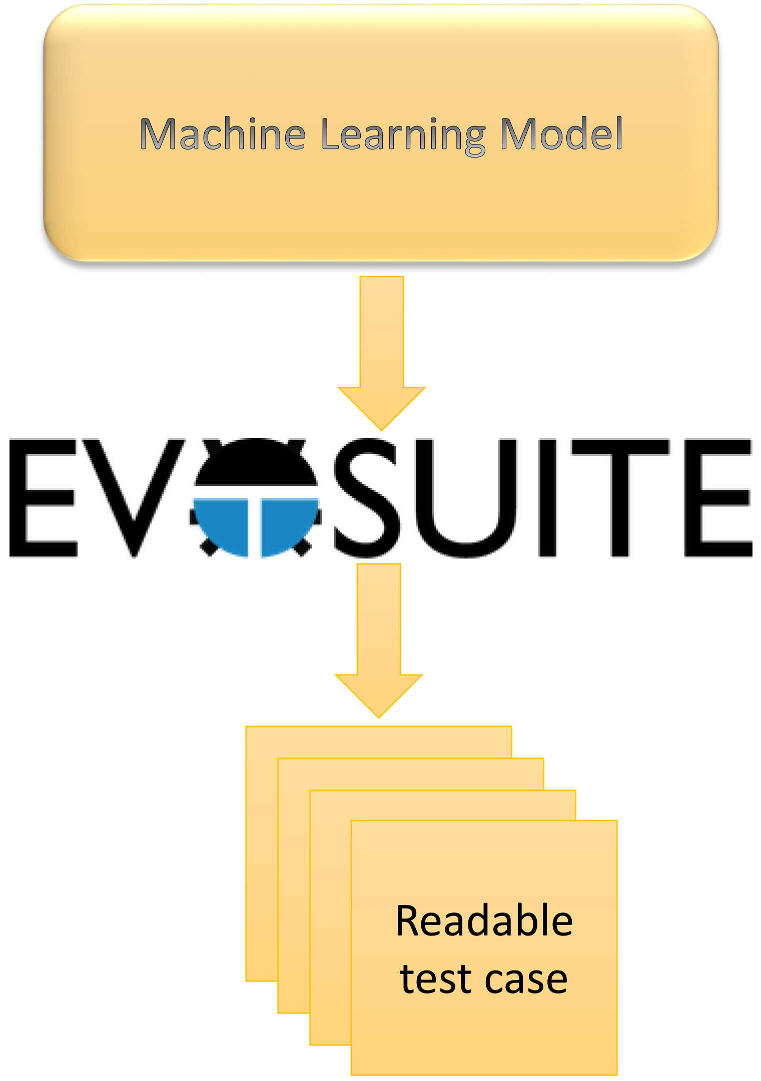
Assertions = 0

String Length = 1

... 99 other features

Feature predictive power

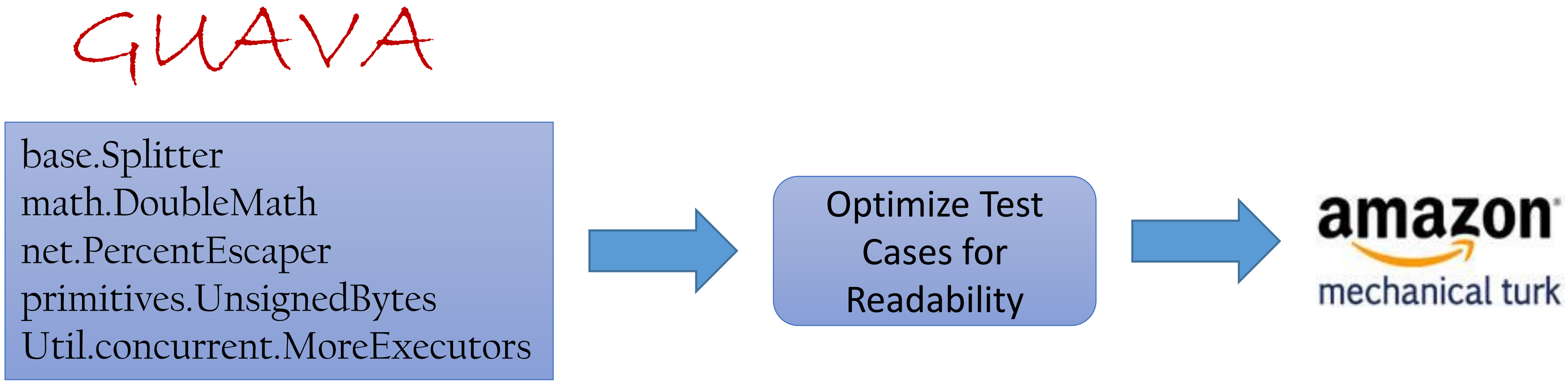




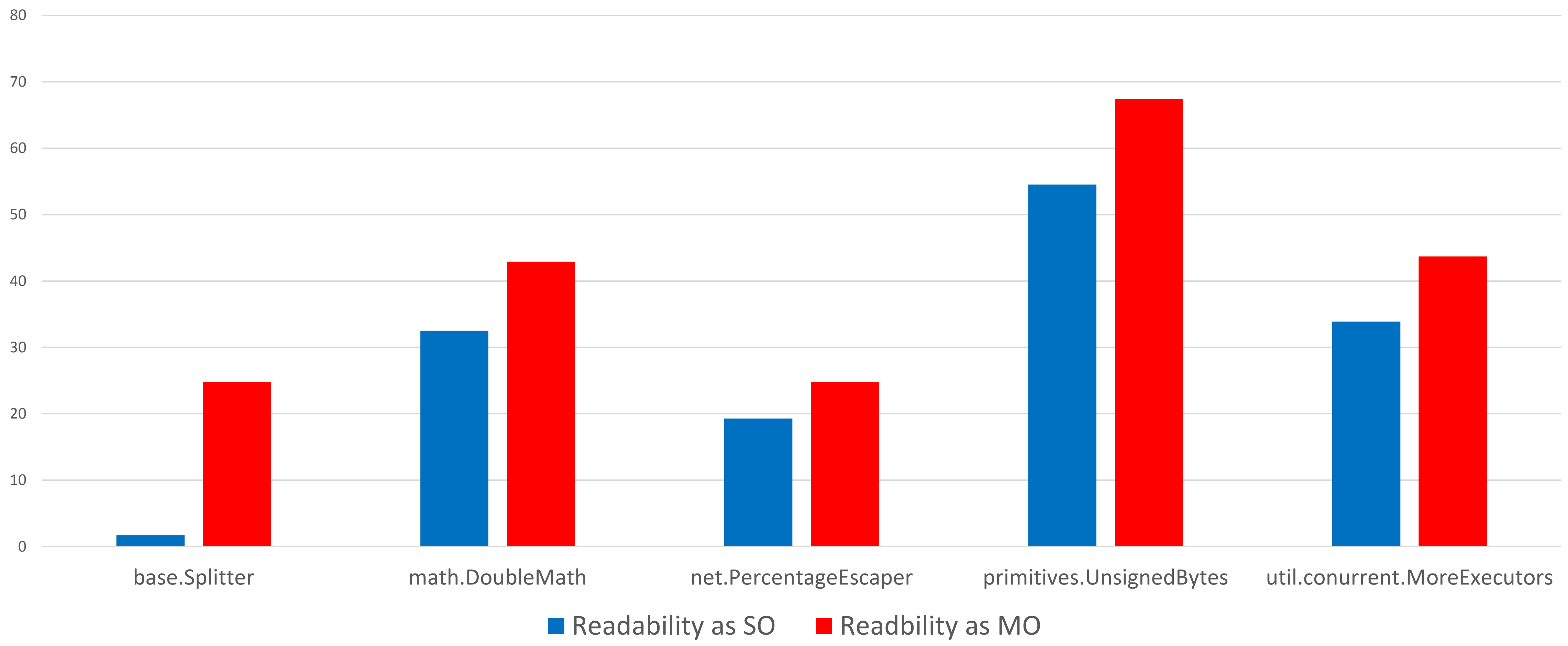
- Readability as secondary objective
  - Individuals with same coverage based fitness are ranked by readability score

- Readability part of multi-objective algorithm
  - NSGA-II including readability score as fitness

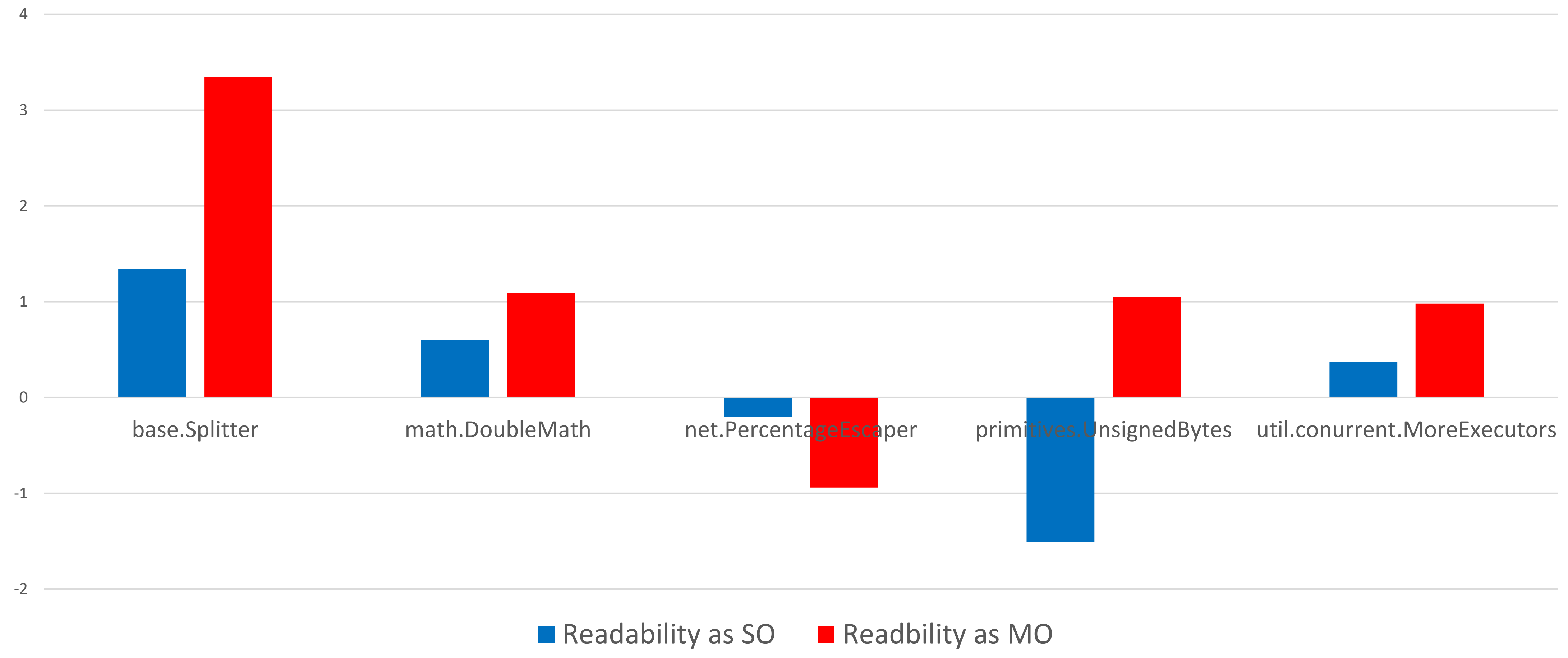
Class selection for test readability evaluation



Readability improvement

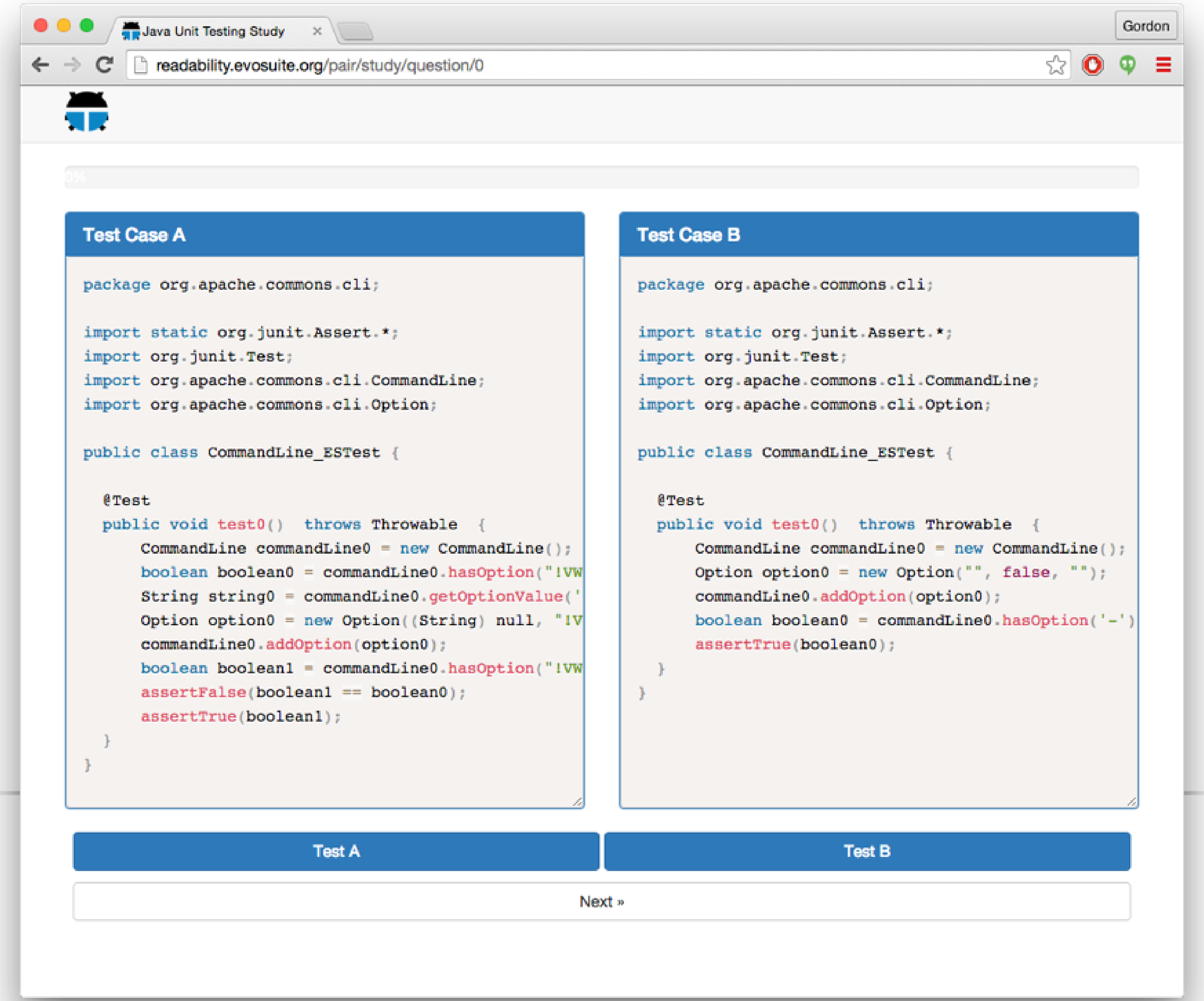


Readability improvement



Do users agree with optimization?

amazon mechanical turk



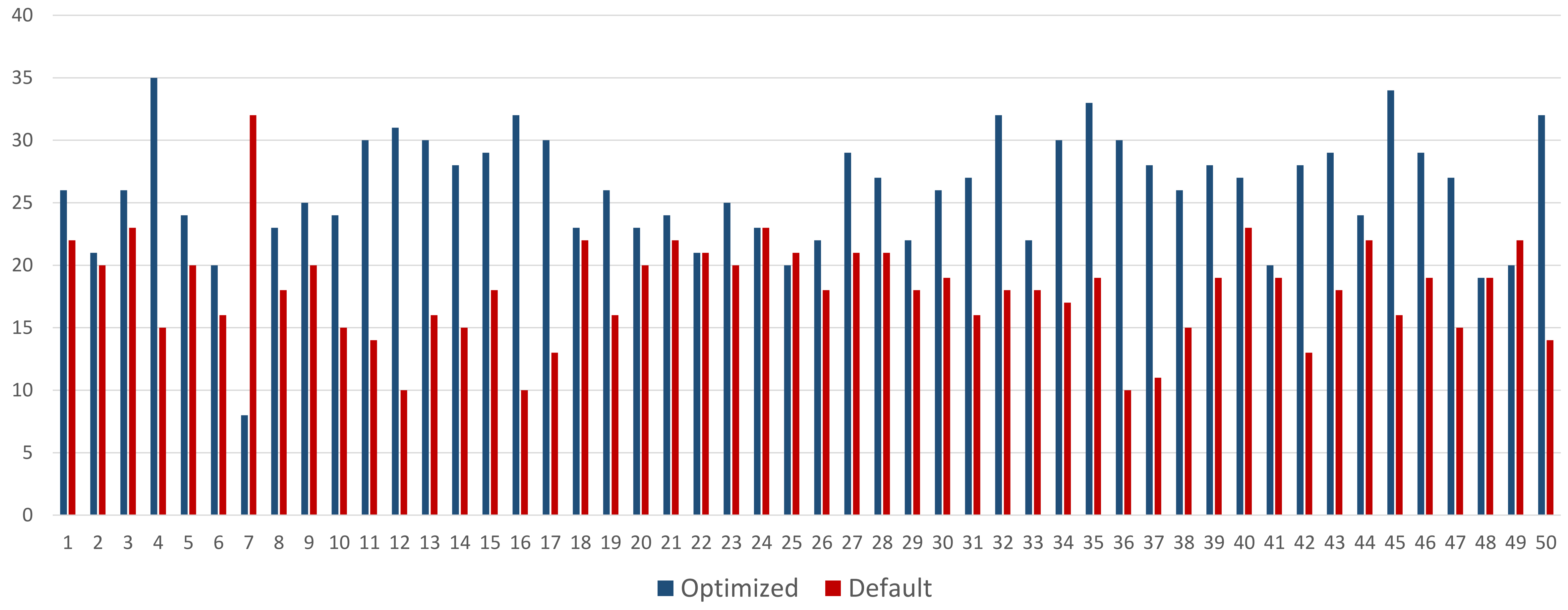
Test Case A

```
package org.apache.commons.cli;

import static org.junit.Assert.*;
import org.junit.Test;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.Option;

public class CommandLine_ESTest {

  @Test
  public void test0()  throws Throwable  {
      CommandLine commandLine0 = new CommandLine();
      boolean boolean0 = commandLine0.hasOption("!VW
      String string0 = commandLine0.getOptionValue('
      Option option0 = new Option((String) null, "!V
      commandLine0.addOption(option0);
      boolean boolean1 = commandLine0.hasOption("!VW
      assertFalse(boolean1 == boolean0);
      assertTrue(boolean1);
  }
}
```

Test Case B

```
package org.apache.commons.cli;

import static org.junit.Assert.*;
import org.junit.Test;
import org.apache.commons.cli.CommandLine;
import org.apache.commons.cli.Option;

public class CommandLine_ESTest {

  @Test
  public void test0()  throws Throwable  {
      CommandLine commandLine0 = new CommandLine();
      Option option0 = new Option("", false, "");
      commandLine0.addOption(option0);
      boolean boolean0 = commandLine0.hasOption('-')
      assertTrue(boolean0);
  }
}
```

Do users agree with optimization?



Readability post-processing

- For a test *t* = <s1, s2,…, si> with coverage goal *c* we:
  - iterate over the statements in the test from the last to the first statement
  - For each statement we determine the possible set of replacement statements
  - For each candidate replacement t’ we determine if it still satisfies *c*
  - Tests are then sorted based on their readability score

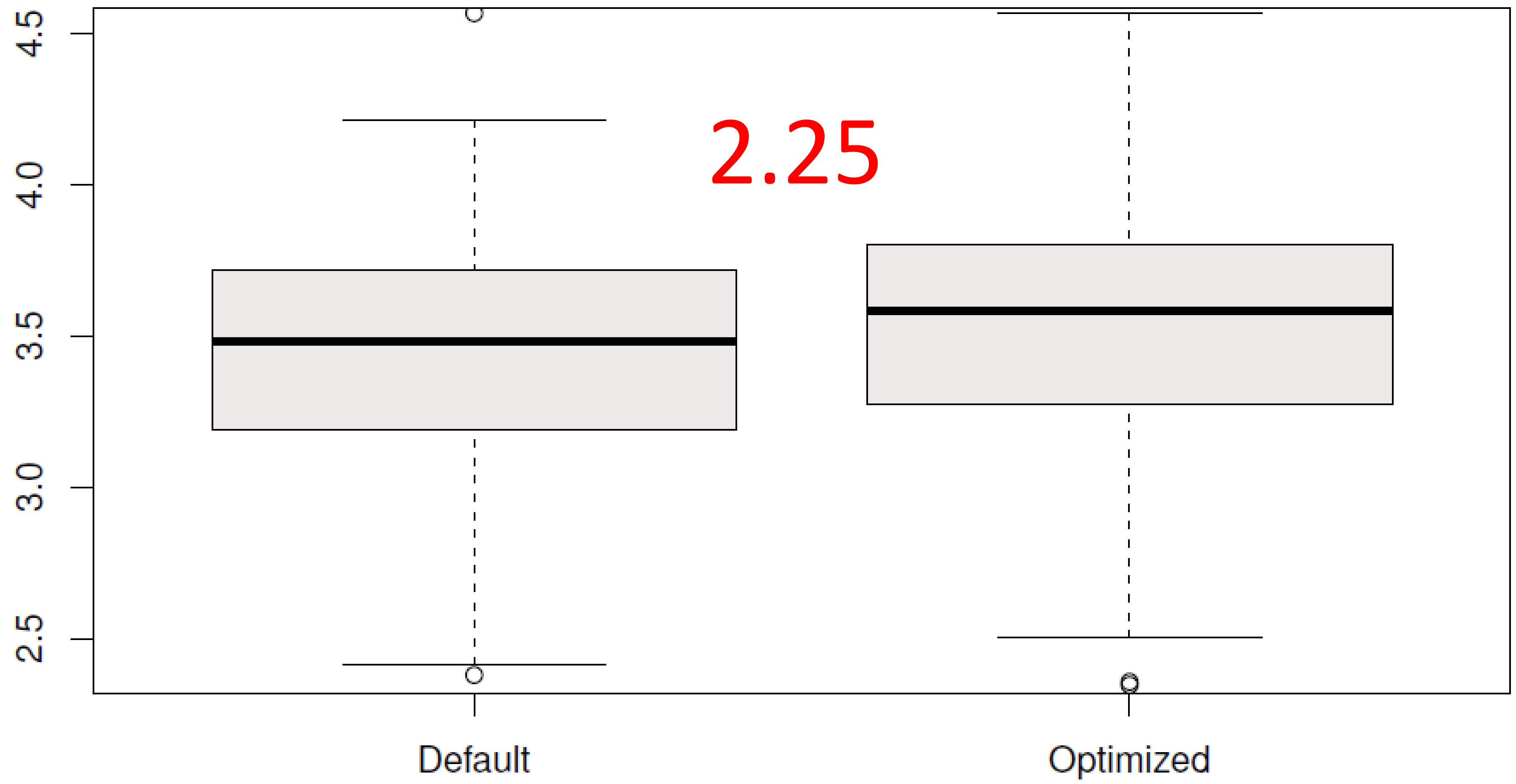
```
Foo foo = new Foo();
Bar bar = new Bar(“Some parameter”, 17);
foo.setBar(bar);
assertTrue(foo.isBar());
```

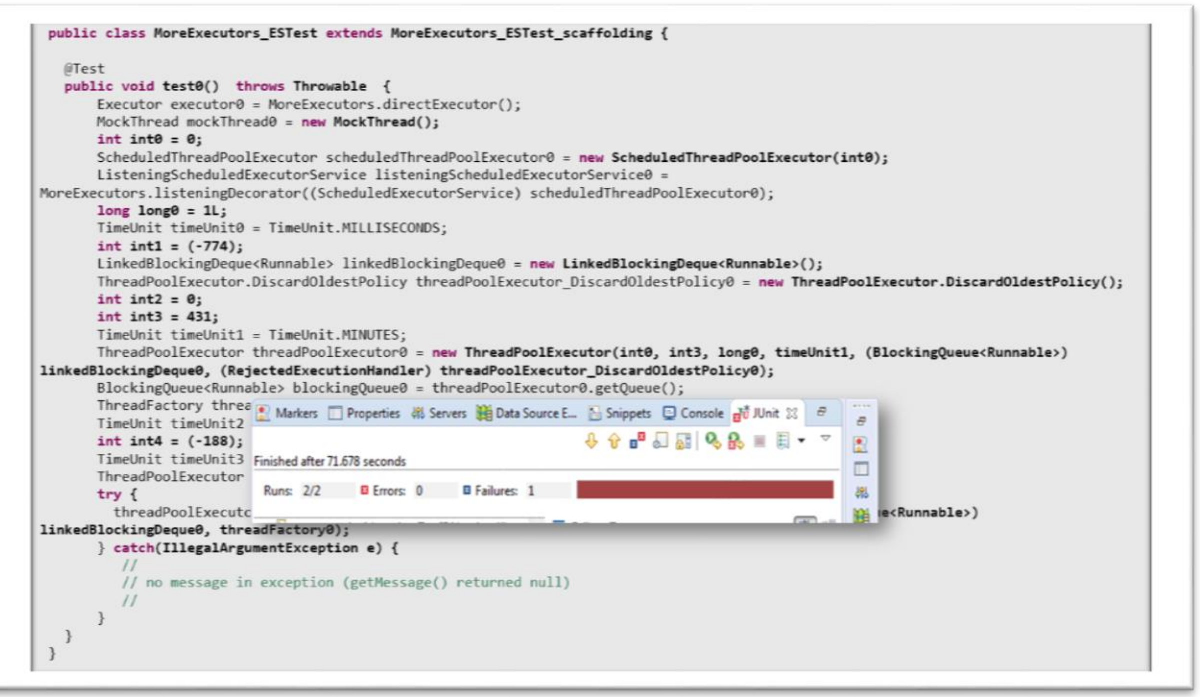
```
Foo foo = new Foo();
Bar bar = new Bar();
foo.setBar(bar);
assertTrue(foo.isBar());
```

```
Bar bar = new Bar();
assertFalse(foo.isBar())
```

Test Suite Generation

- Test suites for all 359 top-level, public classes in Guava

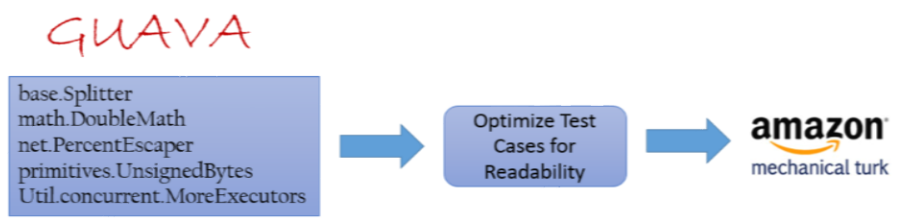




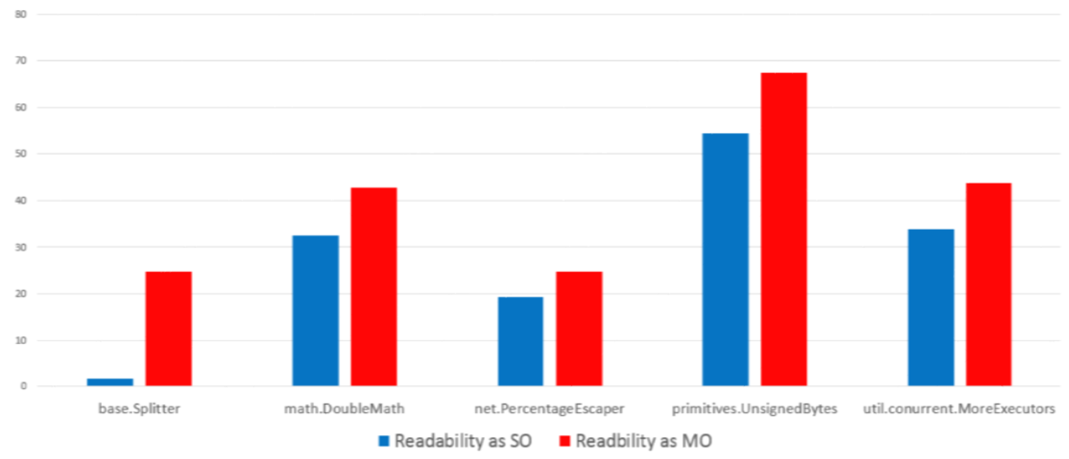
```
public class MoreExecutors_ESTest extends MoreExecutors_ESTest_scaffolding {

  @Test
  public void test0()  throws Throwable  {
      Executor executor0 = MoreExecutors.directExecutor();
      MockThread mockThread0 = new MockThread();
      int int0 = 0;
      ScheduledThreadPoolExecutor scheduledThreadPoolExecutor0 = new ScheduledThreadPoolExecutor(int0);
      ListeningScheduledExecutorService listeningScheduledExecutorService0 =
MoreExecutors.listeningDecorator((ScheduledExecutorService) scheduledThreadPoolExecutor0);
      long long0 = 1L;
      TimeUnit timeUnit0 = TimeUnit.MILLISECONDS;
      int int1 = (-774);
      LinkedBlockingDeque<Runnable> linkedBlockingDeque0 = new LinkedBlockingDeque<Runnable>();
      ThreadPoolExecutor.DiscardOldestPolicy threadPoolExecutor_DiscardOldestPolicy0 = new ThreadPoolExecutor.DiscardOldestPolicy();
      int int2 = 0;
      int int3 = 431;
      TimeUnit timeUnit1 = TimeUnit.MINUTES;
      ThreadPoolExecutor threadPoolExecutor0 = new ThreadPoolExecutor(int0, int3, long0, timeUnit1, (BlockingQueue<Runnable>)
linkedBlockingDeque0, (RejectedExecutionHandler) threadPoolExecutor_DiscardOldestPolicy0);
      BlockingQueue<Runnable> blockingQueue0 = threadPoolExecutor0.getQueue();
      ThreadFactory threa
      TimeUnit timeUnit2
      int int4 = (-188);
      TimeUnit timeUnit3
      ThreadPoolExecutor
      try {
        threadPoolExecutc                                                             e<Runnable>)
linkedBlockingDeque0, threadFactory0);
      } catch(IllegalArgumentException e) {
         //
         // no message in exception (getMessage() returned null)
         //
      }
  }
}
```

```
Finished after 71.678 seconds
Runs: 2/2    Errors: 0    Failures: 1
```

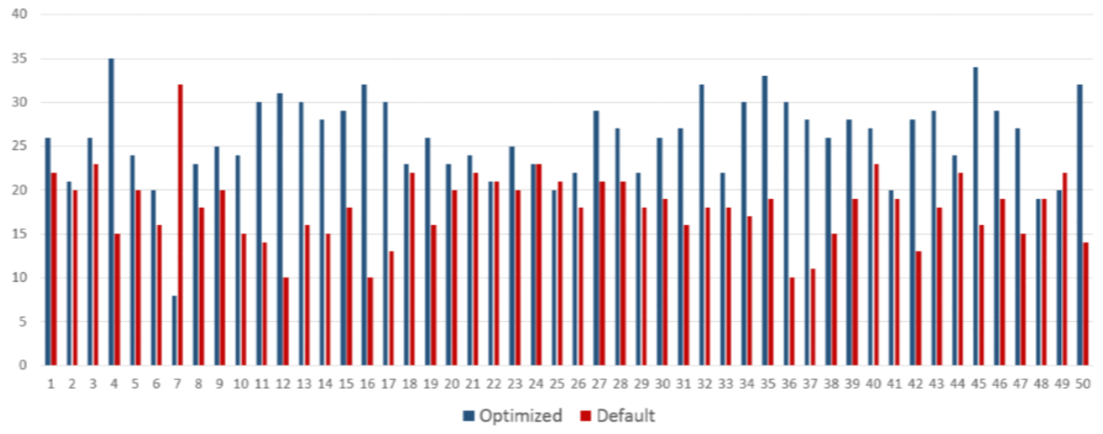
Class selection for test readability evaluation



Readability improvement



Do users agree with optimization?


Optimized
Default