

What is the Vocabulary of Flaky Tests?



**Gustavo
Pinto**



**Breno
Miranda**



**Supun
Dissanayake**



**Marcelo
d'Amorim**



**Christoph
Treude**



**Antonia
Bertolino**



What's a flaky test?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

What's a flaky test?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

Runs: 1/1

 Errors: 0

 Failures: 0

What's a flaky test?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

Runs: 1/1



Errors: 1



Failures: 0

What's a flaky test?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

What's a flaky test?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```


What's a flaky test?

```
public class TestIdentifyEncoder {  
    @Test  
    public void testCodingEmptySrcBuffer() throws Exception {  
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);  
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);  
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();  
  
        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);  
        encoder.write(CodecTestUtils.wrap("stuff"));  
  
        final ByteBuffer empty = ByteBuffer.allocate(100);  
        empty.flip();  
        encoder.write(empty);  
        encoder.write(null);  
        encoder.complete();  
  
        outbuf.flush(channel);  
        final String s = channel.dump(Constants.ASCII);  
  
        Assert.assertTrue(encoder.isCompleted());  
        Assert.assertEquals("stuff", s);  
    }  
}
```


What do we know about flaky tests?

An Empirical Analysis of Flaky Tests

Qingshou Luo, Farah Hanin, Lamyas Elouadi, Darko Marinov
Department of Computer Science, University of Illinois at Urbana-Champaign
Urbana, IL 61801, USA
{qlo2, hanin2, elouadi2, marinov}@illinois.edu

ABSTRACT

Regression testing is a critical part of software development. It checks that software changes do not break existing functionality. An important assumption of regression testing is that test outcomes are deterministic: an unexecuted test is expected to either always pass or always fail for the same code under test. Unfortunately, in practice, some tests either fail or pass inconsistently when executed on the same code under test, a phenomenon known as flaky tests. Flaky tests undermine the regression testing process and are difficult to debug and handle.

We present the first extensive study of flaky tests. We study 16 commits in 201 commits that flake the flake tests in open-source projects. We analyze the root causes, manifest behavior, and developer response strategies that developers use to fix flaky tests. We found that our sample and the population are very similar in terms of root causes and developer response strategies to flaky tests.

Categories and Subject Descriptors: D.2.3 [Software Engineering]: Testing and Debugging

General Terms: Measurement, Reliability

Keywords: Empirical study, flaky tests, non-determinism

1. INTRODUCTION

Regression testing is a critical part of software development. Developers use regression tests to verify that their software changes do not break existing functionality. The goal of running a regression test suite is to find out if the code under test is broken. The outcome is important for developers to take action. If a test fails, developers typically either skip the test or fix the code. If they skip the test, developers know that the code is broken and they need to fix it. If they fix the code, they need to know if the code under test (CUT) is whether the test code is still correct for the code post-fix. In any case, developers need that problem is that a test failure indicates that the code changes have caused a problem in the CUT or the test code.

However, in practice, some tests fail or pass inconsistently on the same code under test. This is known as flaky tests. Flaky tests undermine the regression testing process and are difficult to debug and handle. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. This is a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test.

Qingshou Luo is a Ph.D. student at the University of Illinois at Urbana-Champaign. He is currently working on his thesis titled "An Empirical Analysis of Flaky Tests". He can be reached at qlo2@illinois.edu.

Farah Hanin is a Ph.D. student at the University of Illinois at Urbana-Champaign. She is currently working on her thesis titled "An Empirical Analysis of Flaky Tests". She can be reached at hanin2@illinois.edu.

Lamyas Elouadi is a Ph.D. student at the University of Illinois at Urbana-Champaign. She is currently working on her thesis titled "An Empirical Analysis of Flaky Tests". She can be reached at elouadi2@illinois.edu.

Darko Marinov is a Professor at the University of Illinois at Urbana-Champaign. He is currently working on his research titled "An Empirical Analysis of Flaky Tests". He can be reached at marinov@illinois.edu.

Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. This is a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test.

Flaky tests are not only problematic but also indicative of larger problems in the development process. They point to a big and long-standing problem in general: how to deal with the fact that the only specific numbers we could obtain are that the TMR (Test Manifestation Rate) of a flaky test is a measure of how often it fails. In the past, this was a measure of 100% (4/4) test failures were caused by flaky tests.

The current approach to handle flaky tests is to ignore them. The most common approach is to run a flaky test on a single machine and if it passes, it is considered as passing, even if it fails in some other runs. For example, in Google, a flaky test is considered as passing if it passes on one of the machines it is run on. This is a problem because it is not clear on which machine it is run on, and if it is run on one of the machines, it is not clear if it is a flaky test or not. Several open-source testing frameworks like JUnit and TestNG have tried to deal with flaky tests, but they have different approaches [17] and [18].

Another approach would be to remove flaky tests from the test suite or to manually fix them. But this is not a good idea because it is not clear how to identify flaky tests. The current approach is to ignore them. The most common approach is to run a flaky test on a single machine and if it passes, it is considered as passing, even if it fails in some other runs. For example, in Google, a flaky test is considered as passing if it passes on one of the machines it is run on. This is a problem because it is not clear on which machine it is run on, and if it is run on one of the machines, it is not clear if it is a flaky test or not. Several open-source testing frameworks like JUnit and TestNG have tried to deal with flaky tests, but they have different approaches [17] and [18].

Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. This is a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test. Flaky tests are a problem for developers because they can cause a test to fail or pass inconsistently on the same code under test.

- Analyzed 201 commits that likely fix flaky tests in 51 open-source projects.
- Classify the root causes of flaky tests
- Identify approaches that could manifest flaky behavior
- Describe strategies to fix flaky tests

Luo @ FSE 2014

What do we know about flaky tests?

An Empirical Study of Flaky Tests in Android Apps

Sreyas Thorve, Chandini Ereshtha, Xu Meng
Department of Computer Science
Virginia Tech
Blacksburg, Virginia, U.S.
{sreyas, chandini, xrm27}@cs.vt.edu

Abstract—A flaky test is a test that may fail or pass for the same code under testing (CUT). Flaky tests could be harmful to developers because the non-deterministic test outcome is not reliable and developers cannot easily debug the code. In our study, we characterized the root causes and fixing strategies of flaky tests by analyzing thousands of GitHub open source projects, without analyzing any Android app. Due to the popular usage of Android devices and the multitude of interactions of Android apps with third-party software libraries, hardware, networks, and users, we were curious to find if the Android apps manifested unique failure patterns and called for any special resolution for flaky tests as compared to the existing literature.

For this paper, we conducted an empirical study to characterize the flaky tests in Android apps. By studying the root causes and fixing strategies of failures, we aimed to investigate how our proposed characterizations for failures in Android apps varied from prior findings, and whether there are distinctive specific failure patterns. After mining GitHub, we found 39 Android projects contained 77 commits that were related to failures. We identified the root causes of Android apps' failures. We revealed three main causes: *Platform Dependency*, *Program Logic*, and *UI*. Five types of resolution strategies were observed to address the flaky behavior. Many of the commits also addressed conceptual changes to the failures by changing software implementation to various ways. However, there are still 14% commits that simply delayed or removed the flaky tests. Our characterizations provide useful insights for future research on flaky tests of Android apps.

Index Terms—Android, Flaky tests, empirical

1 INTRODUCTION

Flaky tests are the tests that sometimes give non-deterministic outcomes given the same CUT. When a flaky test is executed multiple times, the testing results of some runs can be "passed" while the other runs' results are "failed". As the outcome becomes non-deterministic, developers cannot simply rely on the outcome to decide whether an app is buggy, neither can they easily debug the code because the failure symptoms may not frequently occur. Researchers in the software industry have revealed the massive influence of flaky tests [1], [2]. John Muenz, a senior Google developer, once mentioned in his blog [3] that across three copies of tests, a consistent rate of about 1.5% of all test runs were seen to report a "flaky" result. Almost half of these tests had some level of failures. Unlike other bugs, flaky tests are non-deterministic and also hard to reproduce. Such tests not only developers and testers confused in debugging, time and effort.

As Android devices have become popular, the rapid development and subsequent usage of Android apps requires

developers to heavily rely on testing for software quality assurance purposes. When test cases are flaky, developers cannot reliably test or improve their Android apps before releasing the software product. Consequently, they may distribute the software components with unreliable app functionalities.

Lee et al. conducted an empirical study on flaky tests by analyzing program commits of 54 Apache open source projects and identified the major root causes and solutions of failures [2]. However, they did not analyze any flaky test for Android apps. We believe that the flaky tests in Android apps should be specially studied for these reasons:

- **Platform Fragmentation.** As the rapid evolution of the Android operating system (OS) continues, a great number of Android OS versions are available in the market, causing the problem of Android fragmentation [4]. As a result, apps may behave differently across different Android platforms and manifest flaky behavior.
- **Diverse Interaction.** Android apps usually interact with various software (e.g., third-party libraries), hardware (e.g., phone devices, networks), different screen sizes, and users. Intuitively, the more parties interacting with Android apps, the more likely that they can introduce non-deterministic or uncontrollable behaviors from the external environment, causing instability in Android apps.
- **Simple Implementation.** Compared with desktop and server applications, Android apps are usually small, and implement easy-to-understand functionalities. If we can understand various scenarios for flaky tests (e.g., when using library A on platform B), in the future, we can build static analysis programs and/or techniques to quickly identify such scenarios in new Android apps, and provide actionable advice to address the failures.

Therefore, we conducted an empirical study on flaky tests of Android apps. Specifically, we crawled for Android apps on GitHub and used the keywords "flaky" and "intermittent" to search for flaky test-related commits. By manually examining the reviewed commits, we found 77 commits in 39 Android projects. We analyzed each commit to investigate the root cause and fixing strategy of failures. This study addresses the following research questions (RQs):

- RQ1: What are the common causes for Android failures? Among the revealed causes, we identified five major

• Xxx

Thorve @ ICSME 2018

Is this test **flaky**?

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));



        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

Runs: 1/1  Errors: 0  Failures: 0

```
@Test
public void testCodingEmptySrcBuffer() throws Exception {
    final WritableByteChannelMock channel = new WritableByteChannelMock(64);
    final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
```

Runs: 1/1  Errors: 0  Failures: 0

```
    final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
    encoder.write(CodecTestUtils.wrap("stuff"));
```



```
    final ByteBuffer empty = ByteBuffer.allocate(100);
    empty.flip();
    encoder.write(empty);
    encoder.write(null);
```

Runs: 1/1  Errors: 0  Failures: 0

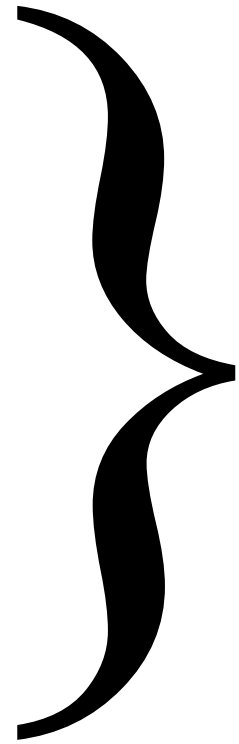
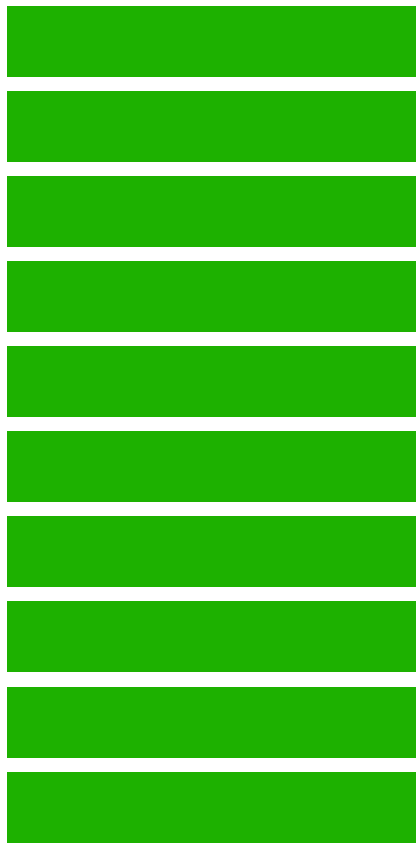
```
    outbuf.flush();
    final String s = channel.dump(Constants.ASCII);
```

Runs: 1/1  Errors: 1  Failures: 0

```
    Assert.assertEquals("stuff", s);
}
```

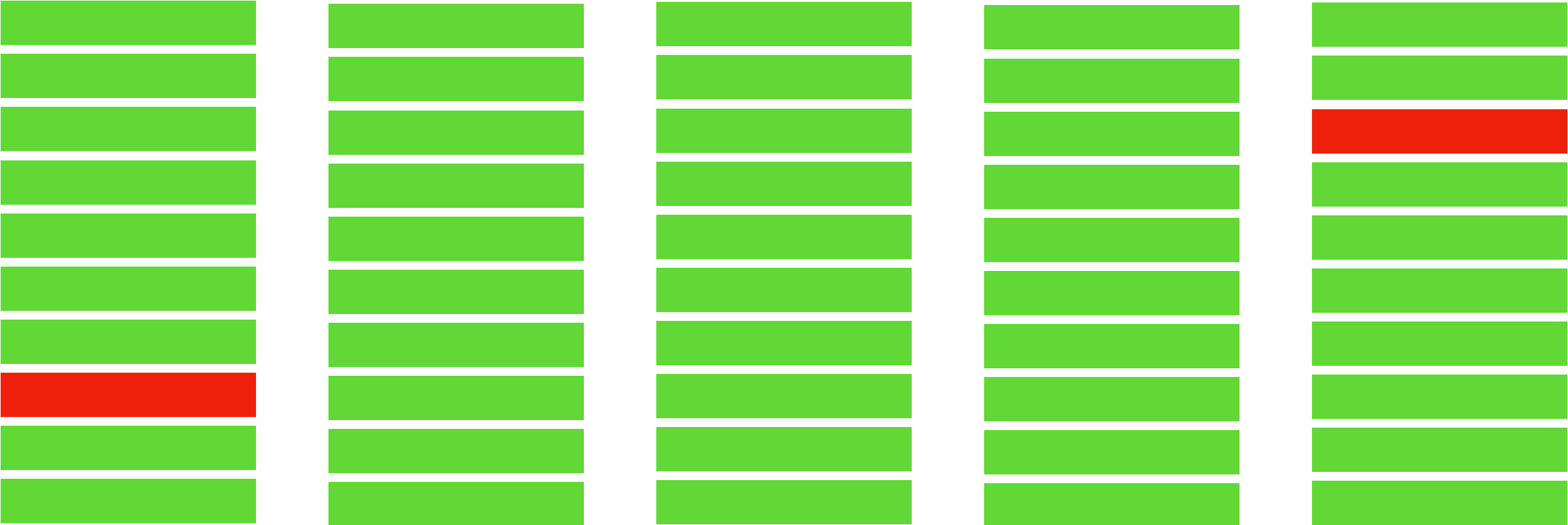
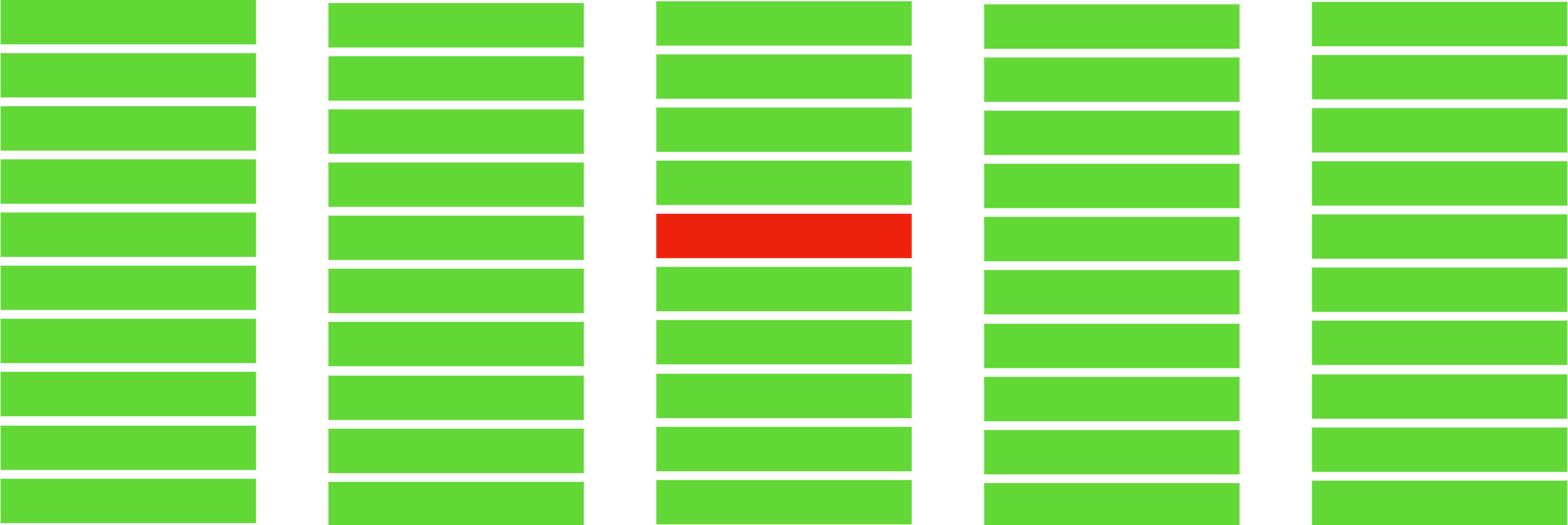
Runs: 1/1  Errors: 0  Failures: 0

Runs: 1/1  Errors: 0  Failures: 0



10 executions







100 executions



GitHub Project

achilles

alluxio

ambari

assertj-core

checkstyle

commons-exec

dropwizard

hadoop

handlebars

hbase

hector

httpcore

jackrabbit-oak

jimfs

logback

ninja

okhttp

oozie

orbit

oryx

spring-boot

togglz

undertow

wro4j

zxing

GitHub Project
achilles
alluxio
ambari
assertj-core
checkstyle
commons-exec
dropwizard
hadoop
handlebars
hbase
hector
httpcore
jackrabbit-oak
jimfs
logback
ninja
okhttp
oozie
orbit
oryx
spring-boot
togglz
undertow
wro4j
zxing

The same studied here

[illegible]

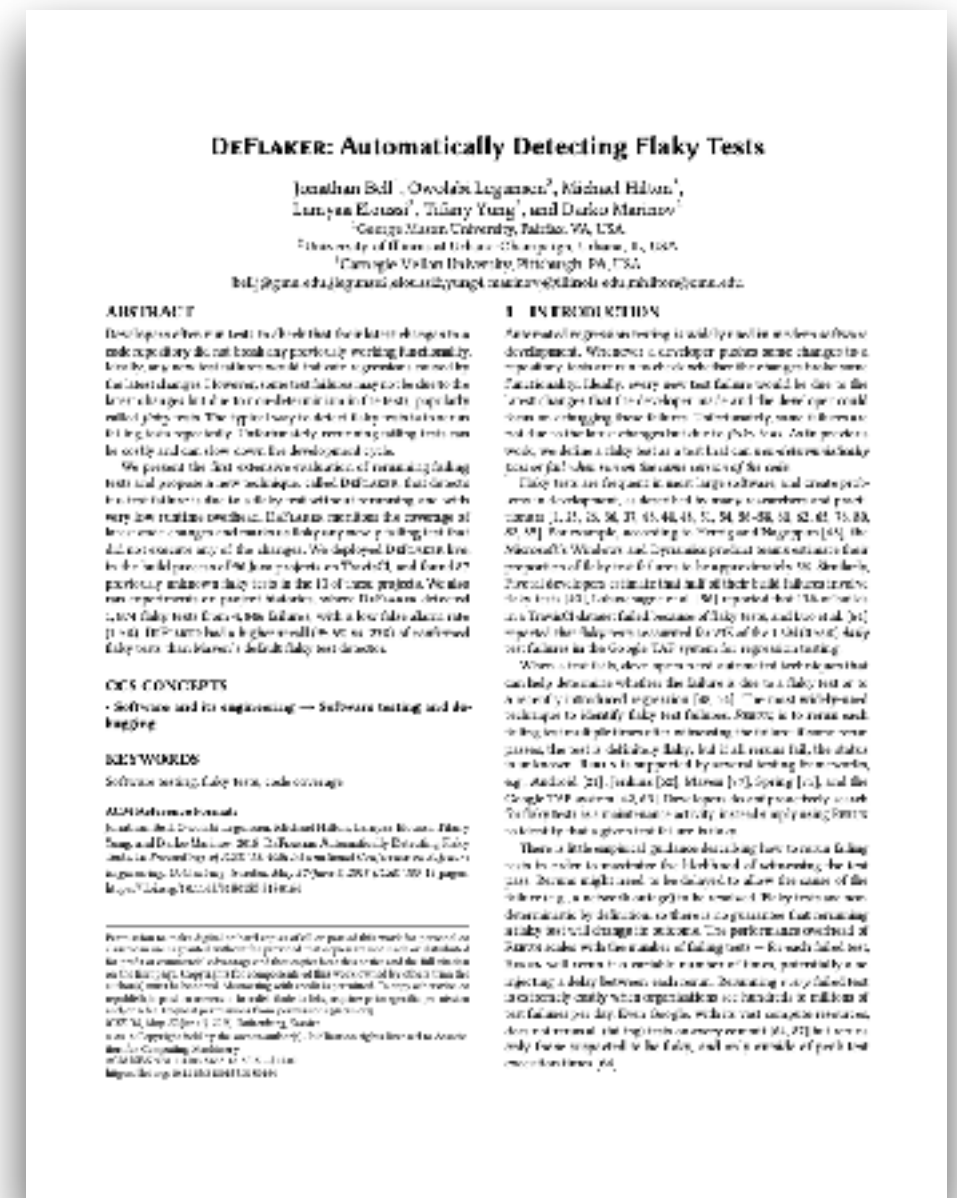
Bell @ ICSE 2018



@gustavopinto

GitHub Project
achilles
alluxio
ambari
assertj-core
checkstyle
commons-exec
dropwizard
hadoop
handlebars
hbase
hector
httpcore
jackrabbit-oak
jimfs
logback
ninja
okhttp
oozie
orbit
oryx
spring-boot
togglz
undertow
wro4j
zxing

The same studied here 

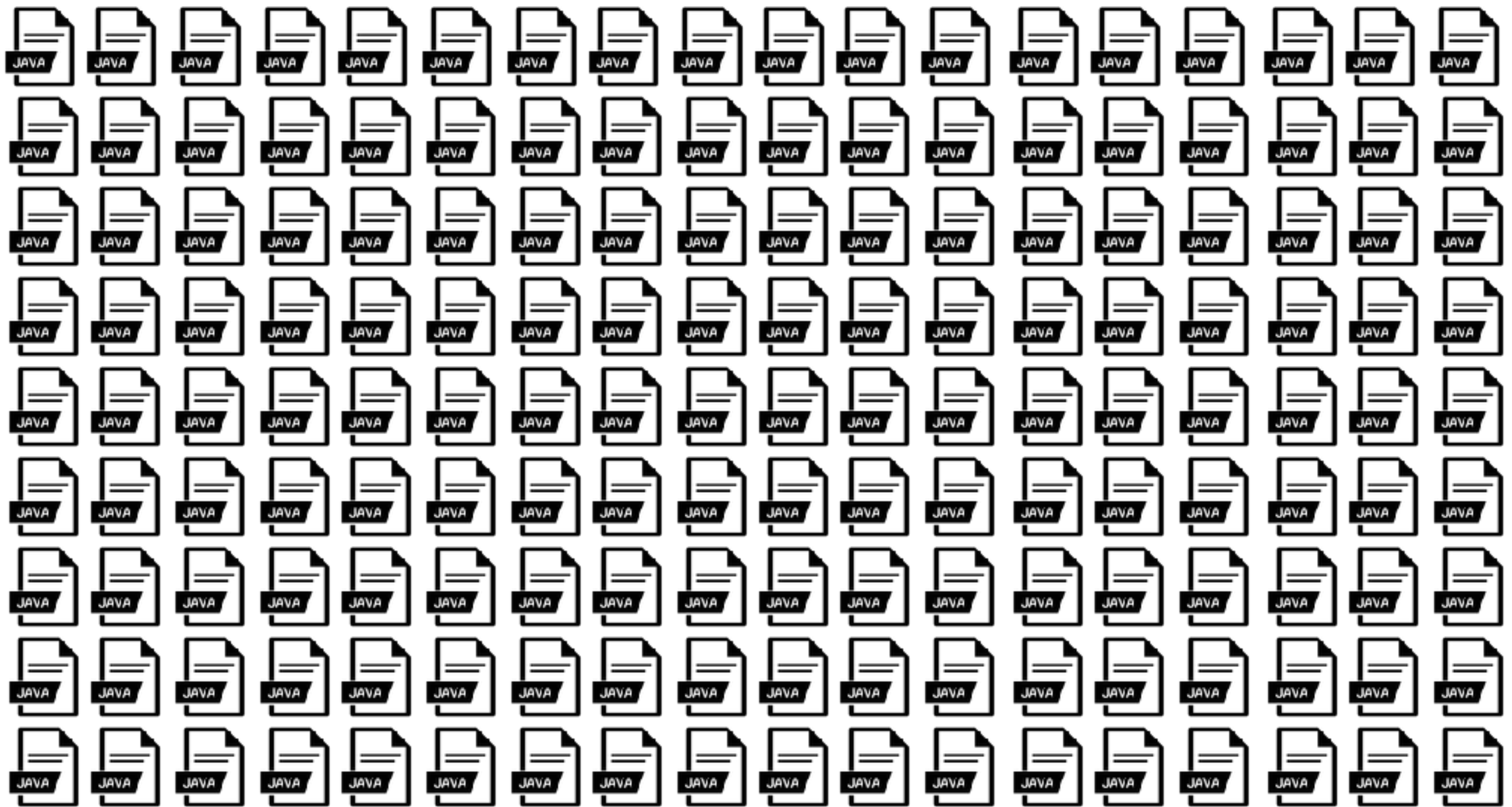


Bell @ ICSE 2018







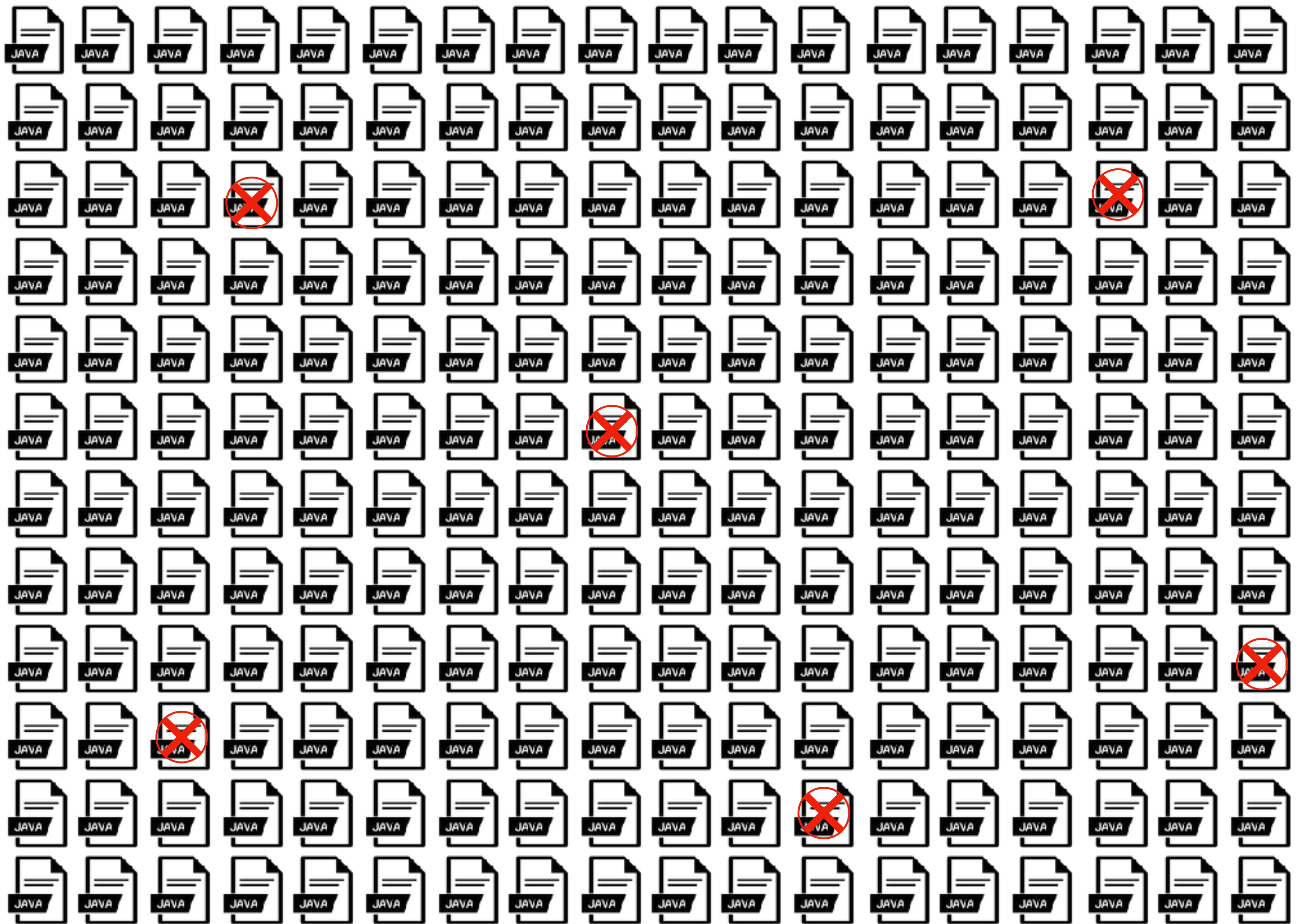


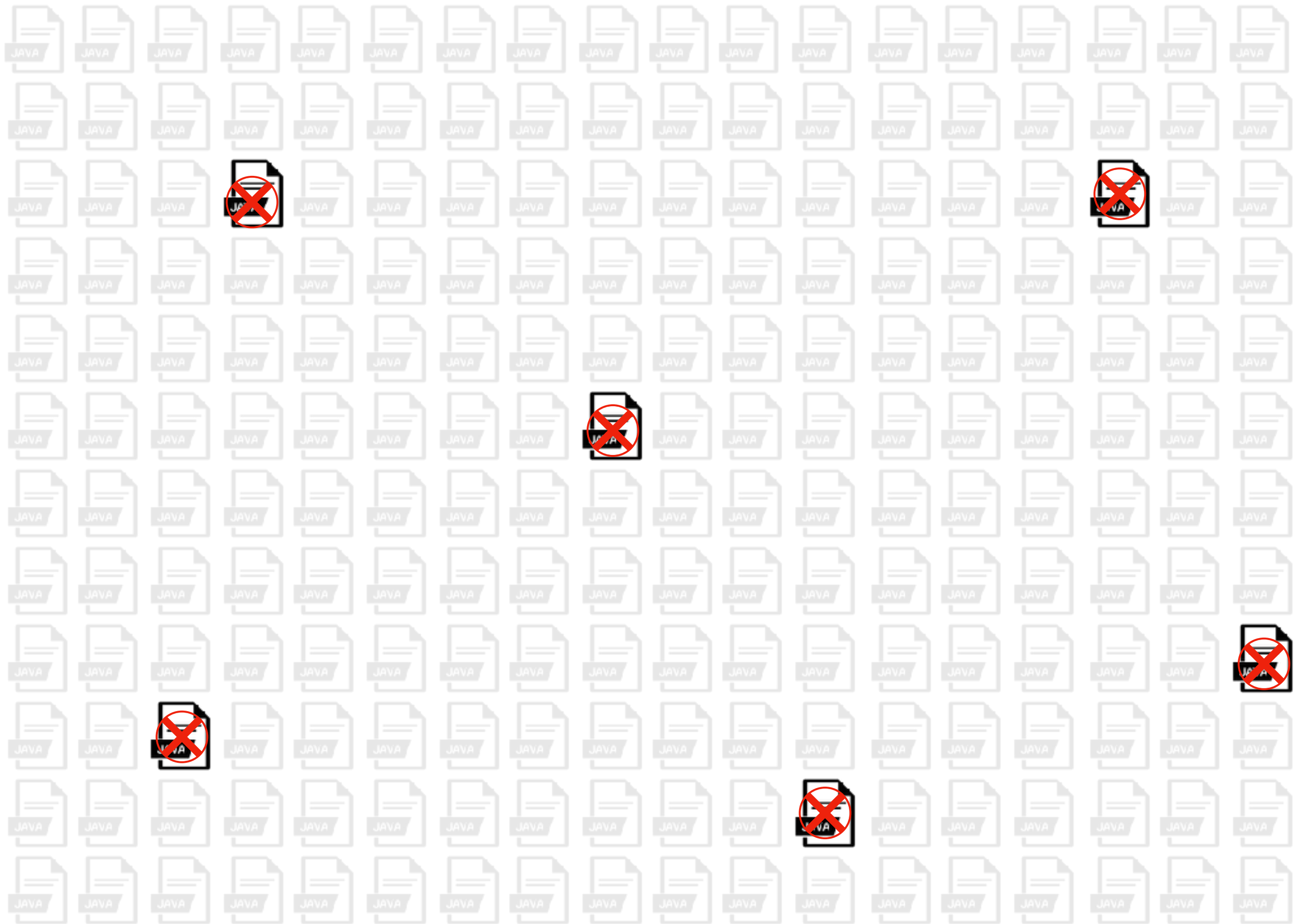
The background of the slide is a repeating pattern of Java file icons. Each icon consists of a document shape with a folded top-right corner and the word "JAVA" in a black box at the bottom. The icons are arranged in a grid that covers the entire slide, with a large yellow rectangle in the center.

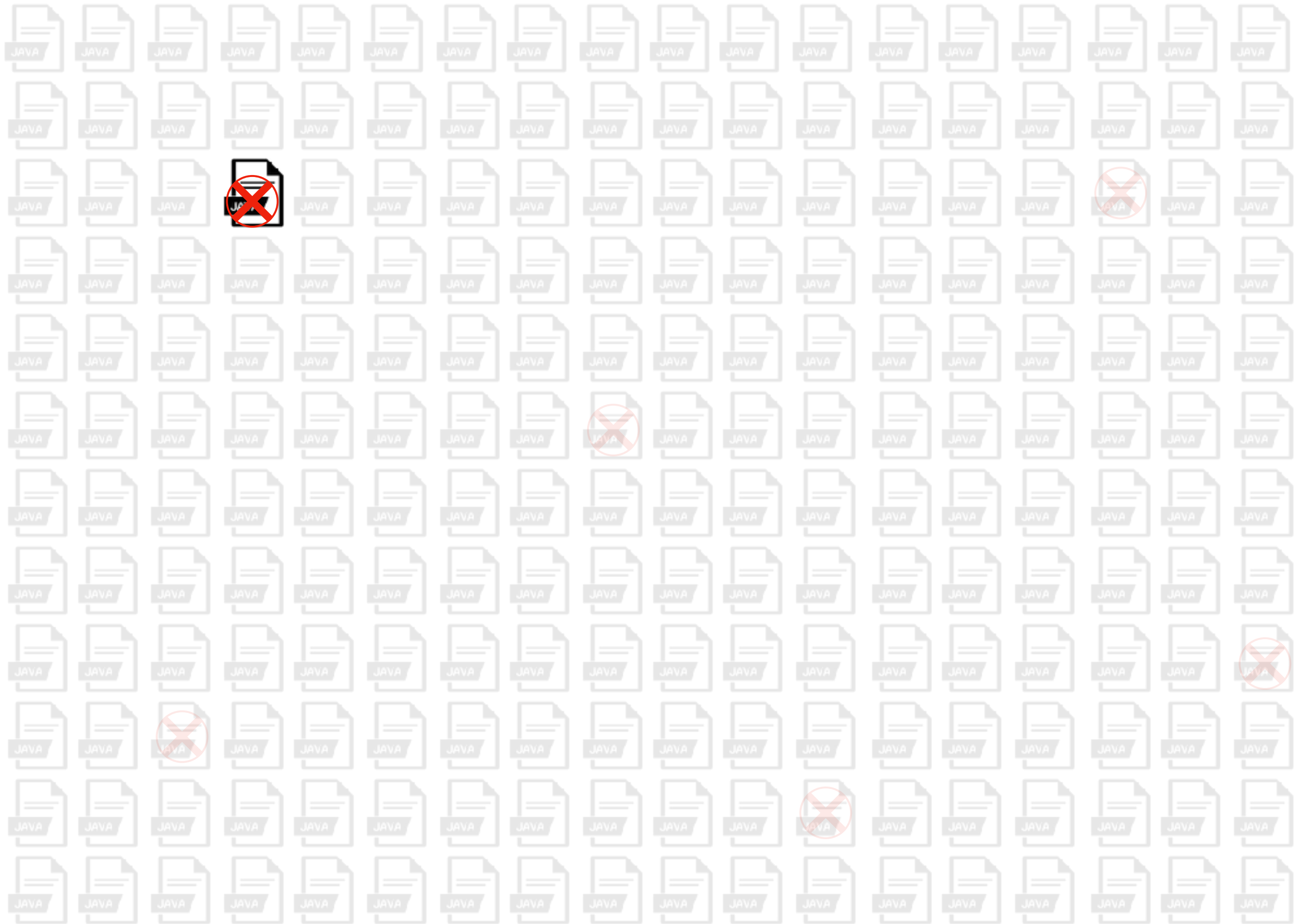
64k test files

The background of the slide is a repeating pattern of Java file icons. Each icon consists of a white document shape with a black border and a black tab on the right side containing the word "JAVA" in white capital letters. These icons are arranged in a grid that covers the entire slide, with a large yellow rectangle in the center.

$64k * 100 =$
 $6.4 \text{ mi executions}$









```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```



```

public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}

```

```
public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}
```

```

public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}

```

```

public class TestIdentifyEncoder {
    @Test
    public void testCodingEmptySrcBuffer() throws Exception {
        final WritableByteChannelMock channel = new WritableByteChannelMock(64);
        final SessionOutputBuffer outbuf = new SessionOutputBufferImpl(1024, 128);
        final HttpTransportMetricsImpl metrics = new HttpTransportMetricsImpl();

        final IdentityEncoder encoder = new IdentityEncoder(channel, outbuf, metrics);
        encoder.write(CodecTestUtils.wrap("stuff"));

        final ByteBuffer empty = ByteBuffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

        outbuf.flush(channel);
        final String s = channel.dump(Constants.ASCII);

        Assert.assertTrue(encoder.isCompleted());
        Assert.assertEquals("stuff", s);
    }
}

```

```

public class test identify encoder {
    @Test
    public void test coding empty src buffer() throws exception {
        final writable byte channel mock channel = new writable byte channel mock(64);
        final session output buffer outbuf = new session output buffer impl(1024, 128);
        final http transport metrics impl metrics = new http transport metrics impl();

        final identity encoder encoder = new identity encoder channel outbuf metrics
encoder.write(codec test utils wrap("stuff"));

        final byte buffer empty = byte buffer.allocate(100);
        empty.flip();
        encoder.write(empty);
        encoder.write(null);
        encoder.complete();

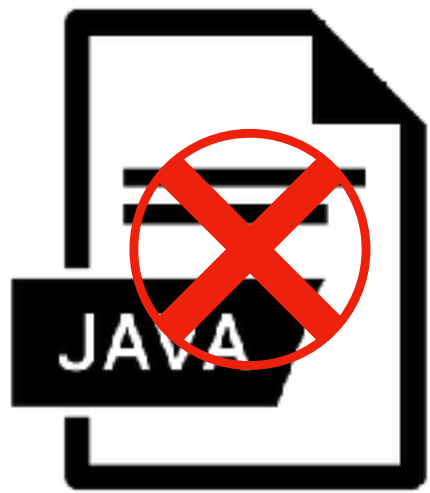
        outbuf.flush(channel);
        final string s = channel.dump(consts.ascii);

        assert.assertTrue(encoder.is completed());
        assertEquals("stuff", s);
    }
}

```



```
test identify encoder coding empty src buffer  
    byte channel mock writable session  
output outbuf session http transport metrics  
impl identity codec utils wrap buffer allocate  
flip write complete flush dump consts ascii is  
    completed assert equals
```



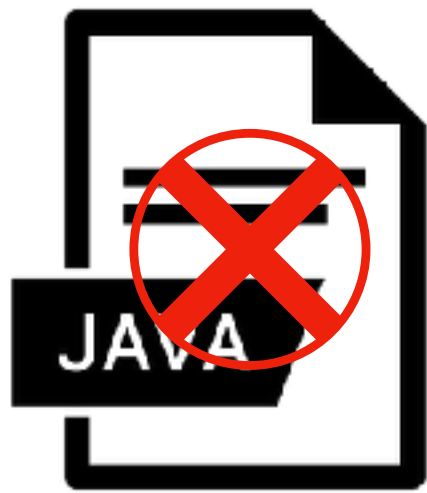
Known Flaky test



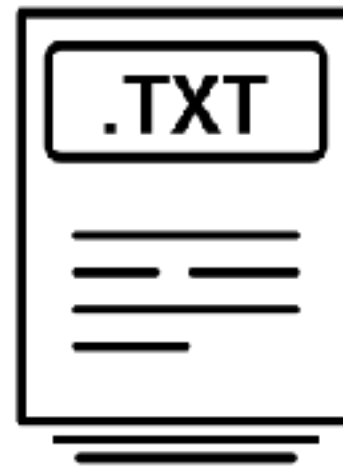
Tokenized Flaky test



**Machine Learning
algorithms**



Known Flaky test



Tokenized Flaky test



**Machine Learning
algorithms**



Weka

Nearest Neighbour

Support Vector
Machine

Decision Tree

Naive Bayes

Random Forest

Implementation available at:
<https://github.com/damorimRG/msr4flakiness/>

RQ1:

How prevalent
are flaky tests?

RQ2:

How accurately can
we **predict** test
flakiness?

RQ3:

What value do
different features add
to the classifier?

RQ4:

Which **test code**
identifiers are strongly
associated with test
flakiness?

RQ1: How prevalent are flaky tests?

Project	# Test	# Flaky	% Flaky
alluxio	3,034	12	0.4
hector	322	40	12.4
jackrabbit-oak	13,193	2	2
okhttp	1,682	19	19
undertow	609	2	2
wro4j	1,158	11	11

25% of the projects
have at least one flaky test

We found
86 flaky tests

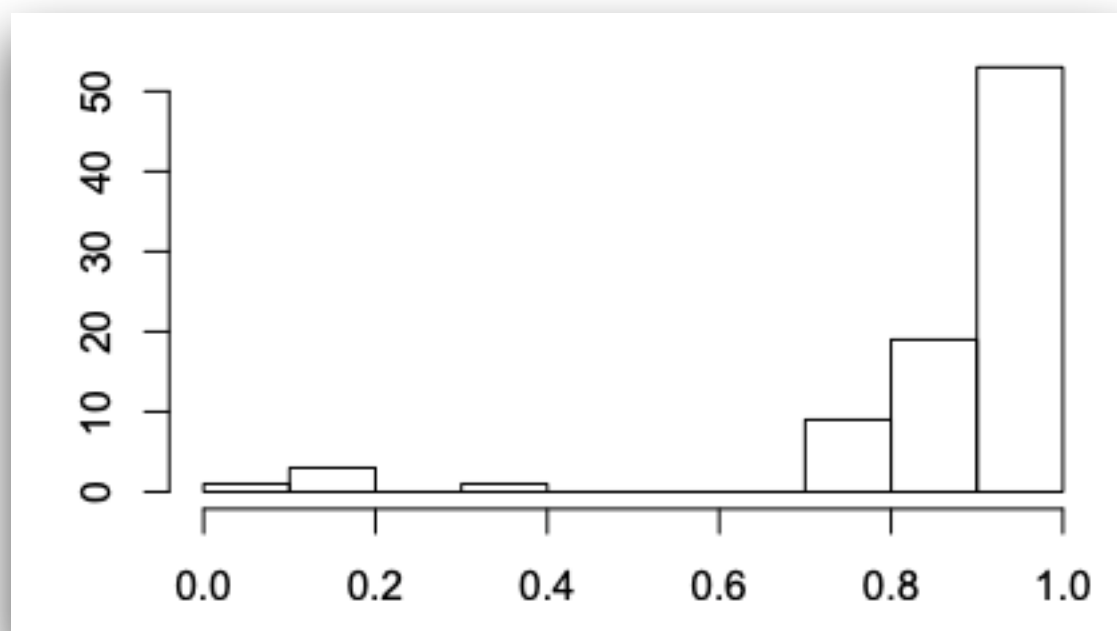
RQ1: How prevalent are flaky tests?

Project	# Test	# Flaky	% Flaky
alluxio	3,034	12	0.4
hector	322	40	12.4
jackrabbit-oak	13,193	2	2
okhttp	1,682	19	19
undertow	609	2	2
wro4j	1,158	11	11

25% of the projects
have at least one flaky test

We found
86 flaky tests

70% (61 out of the 86)
passed more than 90%



RQ2: Can we **predict** flakiness?

ML	Precision	Recall	F1	MCC	AUC
Random Forest	0.99	0.91	0.95	0.90	0.98
Decision Tree	0.98	0.88	0.89	0.77	0.91
Naive Bayes	0.93	0.80	0.86	0.74	0.93
Support Vector	0.93	0.92	0.93	0.85	0.92
Nearest Neighbour	0.97	0.88	0.92	0.85	0.93

Random Forest
achieved best precision

Tuning (e.g., # of trees)
**Had no performance
impact**

RQ3: What's the value of different features

Random Forest

Features	Precision	Recall	F1	MCC	AUC
All features	0.99	0.91	0.95	0.90	0.98
No stemming	0.99	0.91	0.95	0.90	0.98
No Stop W. removal	0.99	0.91	0.95	0.90	0.98
No Lowercasing	0.98	0.91	0.94	0.89	0.98
No Java Keywords	0.99	0.90	0.94	0.89	0.98

RQ3: What's the value of different features

Random Forest

Features	Precision	Recall	F1	MCC	AUC
All features	0.99	0.91	0.95	0.90	0.98
No stemming	0.99	0.91	0.95	0.90	0.98
No Stop W. removal	0.99	0.91	0.95	0.90	0.98
No Lowercasing	0.98	0.91	0.94	0.89	0.98
No Java Keywords	0.99	0.90	0.94	0.89	0.98

Support Vector Machine

Features	Precision	Recall	F1	MCC	AUC
All features	0.93	0.92	0.93	0.85	0.93
No stemming	0.93	0.92	0.93	0.85	0.93
No Stop W. removal	0.93	0.92	0.93	0.85	0.93
No Lowercasing	0.91	0.93	0.92	0.84	0.92
No Java Keywords	0.93	0.92	0.93	0.85	0.93

RQ3: What's the value of different features

Random Forest

Features	Precision	Recall	F1	MCC	AUC
All features	0.99	0.91	0.95	0.90	0.98
No stemming	0.99	0.91	0.95	0.90	0.98
No Stop W. removal	0.99	0.91	0.95	0.90	0.98
No Lowercasing	0.98	0.91	0.94	0.89	0.98
No Java Keywords	0.99	0.90	0.94	0.89	0.98

Support Vector Machine

Features	Precision	Recall	F1	MCC	AUC
All features	0.93	0.92	0.93	0.85	0.93
No stemming	0.93	0.92	0.93	0.85	0.93
No Stop W. removal	0.93	0.92	0.93	0.85	0.93
No Lowercasing	0.91	0.93	0.92	0.84	0.92
No Java Keywords	0.93	0.92	0.93	0.85	0.93

**No performance
impact**

RQ4: What's the **vocabulary**?

Features	inf. gain	Flaky		Non-Flaky	
		# test	# projects	# test	# projects
job	0.20	524	2	4	1
table	0.14	406	4	8	2
id	0.14	552	9	52	4
action	0.13	387	3	8	2
oozie	0.13	274	1	0	0
services	0.13	371	2	7	1
coord	0.11	307	1	0	0
getid	0.11	287	4	1	1
coordinator	0.10	258	1	0	0
xml	0.10	147	2	6	2
workflow	0.09	207	1	0	0
getstatus	0.08	246	2	2	2
record	0.08	296	2	18	1
jpa	0.07	207	2	0	0
jpaservice	0.07	200	1	0	0
service	0.07	367	4	67	3

Common words:
job, table, and action

Many of them associated with
remote tasks
and/or
queue events

GitHub Project
activities
allure
amaven
assuredjpa
chuckstaylor
commons-weather
dropwizard
hadoop
hancicars
hibernate
honor
ITMCore
jackson
jpa
logback
maven
okhttp
osgi
orbit
org
spring-core
spring
undertow
webdriver
zookeeper

