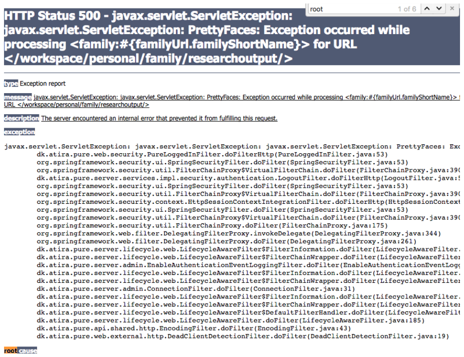
Exceptional Logging

Arie van Deursen
Delft University of Technology

BSR Winterschool, Ede, October 2016

Image credit: goodsonsallterrainlogginginc.com

Big Software on the Run



HTTP Status 500 - javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exception occurred while processing <family:#{familyUrl.familyShortName}> for URL </workspace/personal/family/researchoutput/>

type Exception report

message javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exception occurred while processing <family:#{familyUrl.familyShortName}> f URL </workspace/personal/family/researchoutput/>

description The server encountered an internal error that prevented it from fulfilling this request.

exception

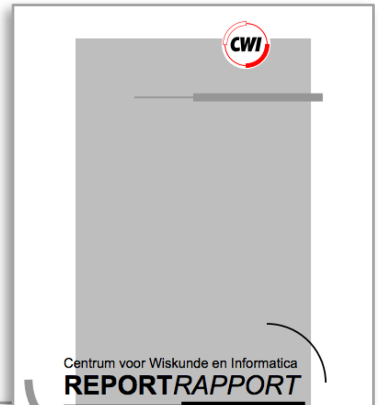
```
javax.servlet.ServletException: javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exc
        dk.atira.pure.web.security.PureLoggedInFilter.doFilterHttp(PureLoggedInFilter.java:53)
        org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
        org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39(
        dk.atira.pure.server.services.impl.security.authentication.LogoutFilter.doFilterHttp(LogoutFilter.java:5
        org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
        org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39(
        org.springframework.security.context.HttpSessionContextIntegrationFilter.doFilterHttp(HttpSessionContext
        org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
        org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39(
        org.springframework.security.util.FilterChainProxy.doFilter(FilterChainProxy.java:175)
        org.springframework.web.filter.DelegatingFilterProxy.invokeDelegate(DelegatingFilterProxy.java:344)
        org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:261)
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilter
        dk.atira.pure.server.admin.EnableAuthenticationEventLoggingFilter.doFilter(EnableAuthenticationEventLog(
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilter
        dk.atira.pure.server.admin.ConnectionFilter.doFilter(ConnectionFilter.java:31)
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilter
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$DefaultFilterHandler.doFilter(LifecycleAwareFilt
        dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter.doFilter(LifecycleAwareFilter.java:185)
        dk.atira.pure.api.shared.http.EncodingFilter.doFilter(EncodingFilter.java:43)
        dk.atira.pure.web.external.http.DeadClientDetectionFilter.doFilter(DeadClientDetectionFilter.java:19)
```

root cause

- Long running (distributed) systems
  - *All situations will occur*
  - *All exceptions will fire*

- How do developers deal with exceptions?
- How do exceptions end up in crashes and issues?
- How do exceptions manifest themselves in log data?
- How can logged exceptions be reproduced?

[Error Handling and Type Checking – 1991]



Centrum voor Wiskunde en Informatica

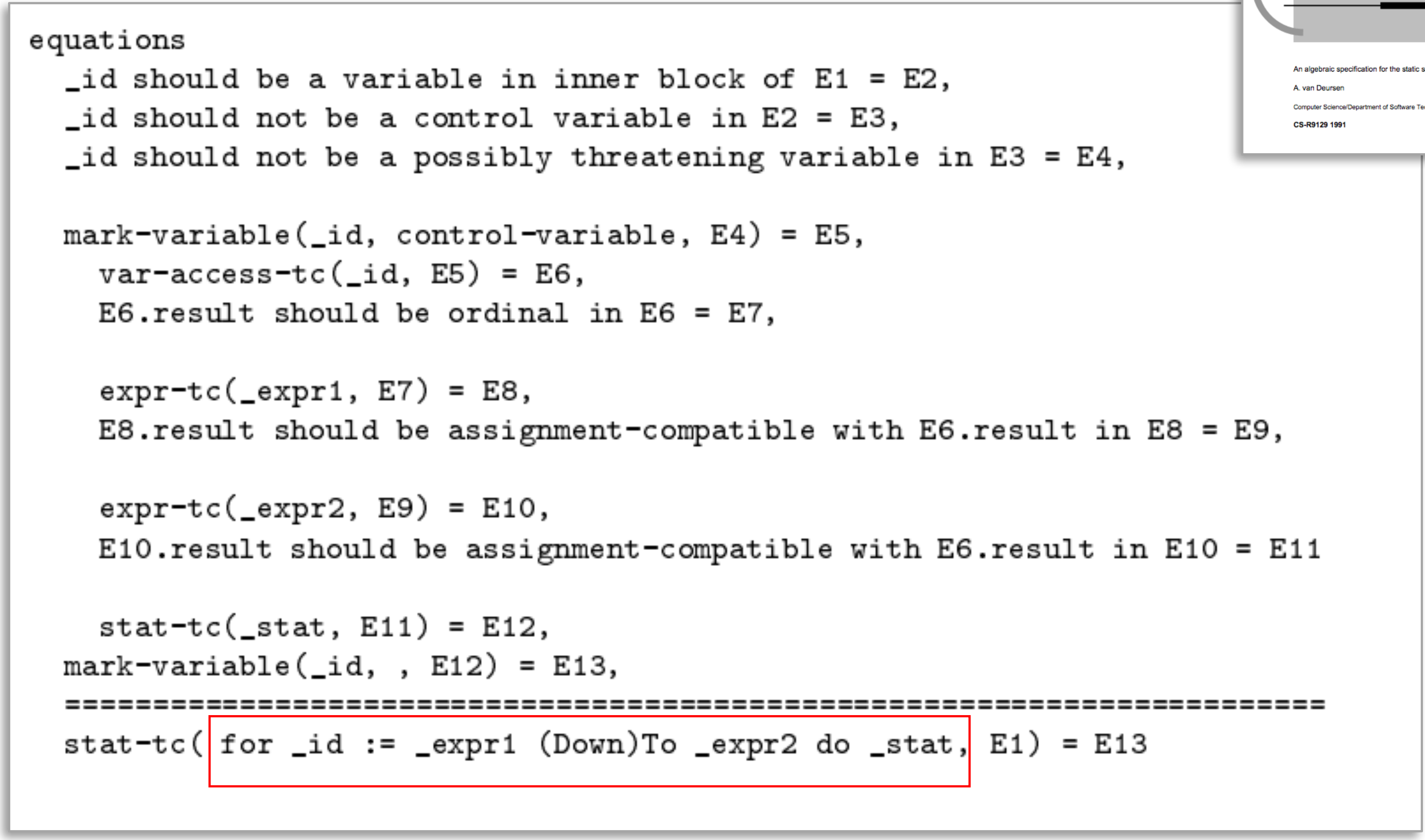
REPORTRAPPORT

An algebraic specification for the static semantics of Pascal

A. van Deursen

Computer Science/Department of Software Technology

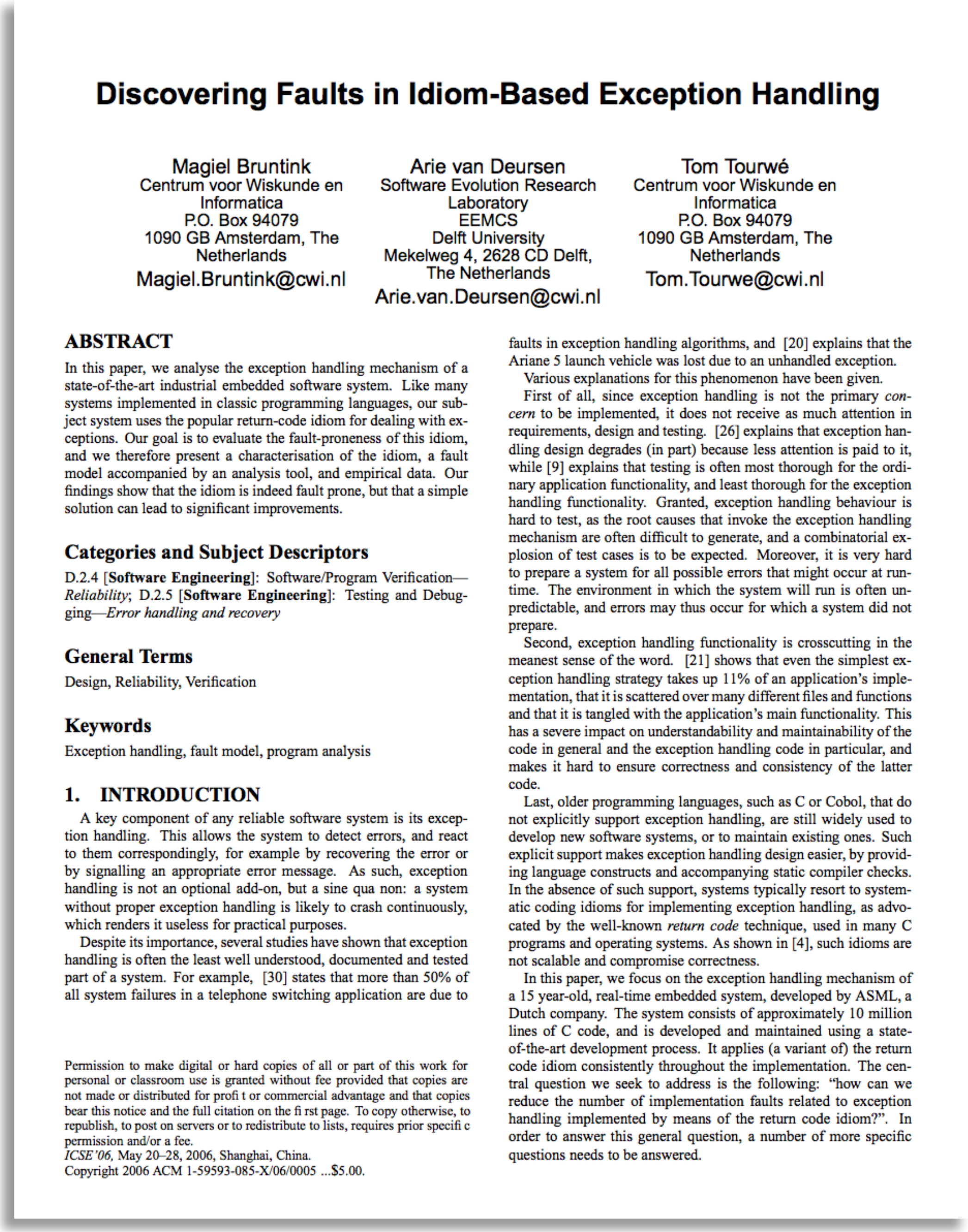
CS-R9129 1991



```
equations
  _id should be a variable in inner block of E1 = E2,
  _id should not be a control variable in E2 = E3,
  _id should not be a possibly threatening variable in E3 = E4,

  mark-variable(_id, control-variable, E4) = E5,
    var-access-tc(_id, E5) = E6,
    E6.result should be ordinal in E6 = E7,

    expr-tc(_expr1, E7) = E8,
    E8.result should be assignment-compatible with E6.result in E8 = E9,

    expr-tc(_expr2, E9) = E10,
    E10.result should be assignment-compatible with E6.result in E10 = E11

    stat-tc(_stat, E11) = E12,
  mark-variable(_id, , E12) = E13,
  ========================================================================
  stat-tc( for _id := _expr1 (Down)To _expr2 do _stat, E1) = E13
```

Shanghai 2006

ICSE



Discovering Faults in Idiom-Based Exception Handling

Magiel Bruntink
Centrum voor Wiskunde en Informatica
P.O. Box 94079
1090 GB Amsterdam, The Netherlands
Magiel.Bruntink@cwi.nl

Arie van Deursen
Software Evolution Research Laboratory
EEMCS
Delft University
Mekelweg 4, 2628 CD Delft, The Netherlands
Arie.van.Deursen@cwi.nl

Tom Tourwé
Centrum voor Wiskunde en Informatica
P.O. Box 94079
1090 GB Amsterdam, The Netherlands
Tom.Tourwe@cwi.nl

ABSTRACT

In this paper, we analyse the exception handling mechanism of a state-of-the-art industrial embedded software system. Like many systems implemented in classic programming languages, our subject system uses the popular return-code idiom for dealing with exceptions. Our goal is to evaluate the fault-proneness of this idiom, and we therefore present a characterisation of the idiom, a fault model accompanied by an analysis tool, and empirical data. Our findings show that the idiom is indeed fault prone, but that a simple solution can lead to significant improvements.

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Reliability*; D.2.5 [**Software Engineering**]: Testing and Debugging—*Error handling and recovery*

General Terms

Design, Reliability, Verification

Keywords

Exception handling, fault model, program analysis

1. INTRODUCTION

A key component of any reliable software system is its exception handling. This allows the system to detect errors, and react to them correspondingly, for example by recovering the error or by signalling an appropriate error message. As such, exception handling is not an optional add-on, but a sine qua non: a system without proper exception handling is likely to crash continuously, which renders it useless for practical purposes.

Despite its importance, several studies have shown that exception handling is often the least well understood, documented and tested part of a system. For example, [30] states that more than 50% of all system failures in a telephone switching application are due to faults in exception handling algorithms, and [20] explains that the Ariane 5 launch vehicle was lost due to an unhandled exception.

Various explanations for this phenomenon have been given.

First of all, since exception handling is not the primary *concern* to be implemented, it does not receive as much attention in requirements, design and testing. [26] explains that exception handling design degrades (in part) because less attention is paid to it, while [9] explains that testing is often most thorough for the ordinary application functionality, and least thorough for the exception handling functionality. Granted, exception handling behaviour is hard to test, as the root causes that invoke the exception handling mechanism are often difficult to generate, and a combinatorial explosion of test cases is to be expected. Moreover, it is very hard to prepare a system for all possible errors that might occur at runtime. The environment in which the system will run is often unpredictable, and errors may thus occur for which a system did not prepare.

Second, exception handling functionality is crosscutting in the meanest sense of the word. [21] shows that even the simplest exception handling strategy takes up 11% of an application's implementation, that it is scattered over many different files and functions and that it is tangled with the application's main functionality. This has a severe impact on understandability and maintainability of the code in general and the exception handling code in particular, and makes it hard to ensure correctness and consistency of the latter code.

Last, older programming languages, such as C or Cobol, that do not explicitly support exception handling, are still widely used to develop new software systems, or to maintain existing ones. Such explicit support makes exception handling design easier, by providing language constructs and accompanying static compiler checks. In the absence of such support, systems typically resort to systematic coding idioms for implementing exception handling, as advocated by the well-known *return code* technique, used in many C programs and operating systems. As shown in [4], such idioms are not scalable and compromise correctness.

In this paper, we focus on the exception handling mechanism of a 15 year-old, real-time embedded system, developed by ASML, a Dutch company. The system consists of approximately 10 million lines of C code, and is developed and maintained using a state-of-the-art development process. It applies (a variant of) the return code idiom consistently throughout the implementation. The central question we seek to address is the following: "how can we reduce the number of implementation faults related to exception handling implemented by means of the return code idiom?". In order to answer this general question, a number of more specific questions needs to be answered.

ICSE'06, May 20–28, 2006, Shanghai, China.
Copyright 2006 ACM 1-59593-085-X/06/0005 ...$5.00.

The Error Linking Concern

- Proper error logging essential for
  - Activating correct recovery procedure
  - Making sure repair engineer can take proper action upon crash
  - Directly affects *repair time,* hence *uptime.*
- Different error codes are *linked* back to *root cause*
- Explicitly described “error linking idiom”

Error Linking

```
int queue_add(CCQU_queue  *queue, void *item_data, bool front) {
  int  r = OK;
  …
  if ((r == OK) && (queue == (CCQU_queue *) NULL)) {
     r = CCXA_PARAMETER_ERR;
     ERXA_LOG(r, 0);
  }
  …
  if (r == OK) {
     r = PLXAmem_malloc(sizeof(CCQU_queue_item), (void **) &qi);
     if (r != OK) {
        ERXA_LOG(r, 0);
        ERXA_LOG(CCXA_MEMORY_ERR, r);
        r = CCXA_MEMORY_ERR;
     }
  }

  if (r == OK) {
     ...
  }

  return r;
}
```

Initialization

Root error

Error logging

Skipping

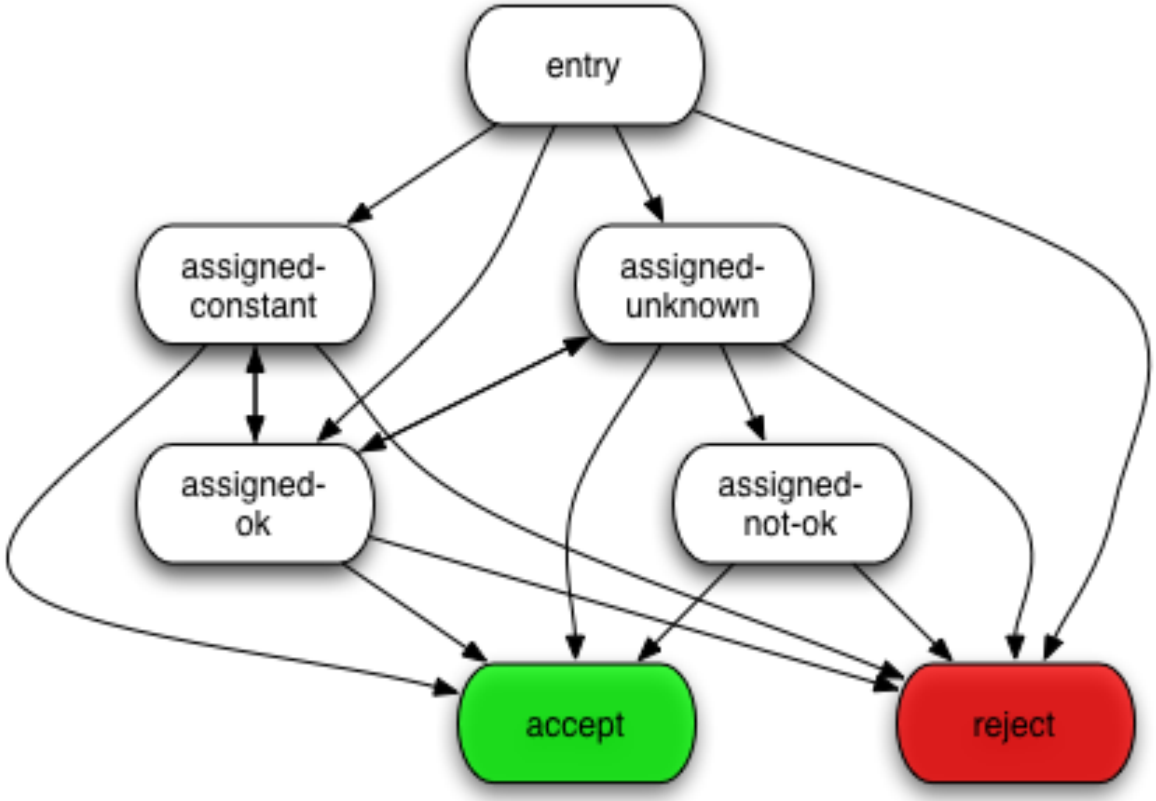
Linking

Propagation

Error Linking Quality

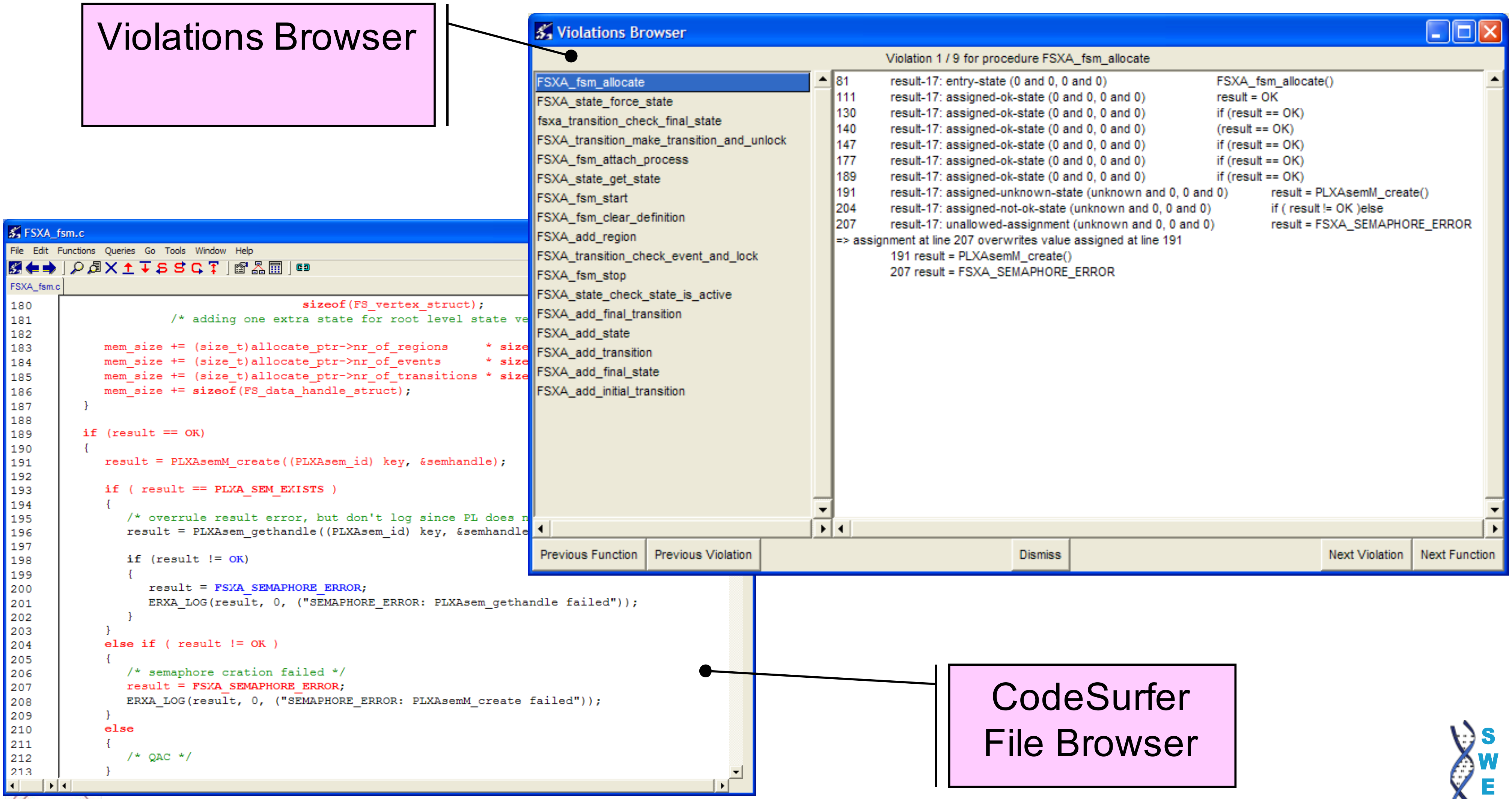
- Fault model:
  - Wrong error variable returned,
  - assigned and logged value mismatch,
  - not linked to previous value,
  - omit guard, ...
- Can be identified using analysis of *program dependence graph*
- For each path find out whether error value is properly set, checked, logged and returned

Static Analysis in SMELL



State
Machine for
Error
Linking and
Logging

Violation reports



Violation 1 / 9 for procedure FSXA_fsm_allocate

FSXA_fsm_allocate
FSXA_state_force_state
fsxa_transition_check_final_state
FSXA_transition_make_transition_and_unlock
FSXA_fsm_attach_process
FSXA_state_get_state
FSXA_fsm_start
FSXA_fsm_clear_definition
FSXA_add_region
FSXA_transition_check_event_and_lock
FSXA_fsm_stop
FSXA_state_check_state_is_active
FSXA_add_final_transition
FSXA_add_state
FSXA_add_transition
FSXA_add_final_state
FSXA_add_initial_transition

```
81      result-17: entry-state (0 and 0, 0 and 0)                         FSXA_fsm_allocate()
111     result-17: assigned-ok-state (0 and 0, 0 and 0)                   result = OK
130     result-17: assigned-ok-state (0 and 0, 0 and 0)                   if (result == OK)
140     result-17: assigned-ok-state (0 and 0, 0 and 0)                   (result == OK)
147     result-17: assigned-ok-state (0 and 0, 0 and 0)                   if (result == OK)
177     result-17: assigned-ok-state (0 and 0, 0 and 0)                   if (result == OK)
189     result-17: assigned-ok-state (0 and 0, 0 and 0)                   if (result == OK)
191     result-17: assigned-unknown-state (unknown and 0, 0 and 0)            result = PLXAsemM_create()
204     result-17: assigned-not-ok-state (unknown and 0, 0 and 0)             if ( result != OK )else
207     result-17: unallowed-assignment (unknown and 0, 0 and 0)              result = FSXA_SEMAPHORE_ERROR
=> assignment at line 207 overwrites value assigned at line 191
        191 result = PLXAsemM_create()
        207 result = FSXA_SEMAPHORE_ERROR
```

```
180                          sizeof(FS_vertex_struct);
181                /* adding one extra state for root level state ve
182
183       mem_size += (size_t)allocate_ptr->nr_of_regions     * size
184       mem_size += (size_t)allocate_ptr->nr_of_events      * size
185       mem_size += (size_t)allocate_ptr->nr_of_transitions * size
186       mem_size += sizeof(FS_data_handle_struct);
187    }
188
189    if (result == OK)
190    {
191       result = PLXAsemM_create((PLXAsem_id) key, &semhandle);
192
193       if ( result == PLXA_SEM_EXISTS )
194       {
195          /* overrule result error, but don't log since PL does n
196          result = PLXAsem_gethandle((PLXAsem_id) key, &semhandle
197
198          if (result != OK)
199          {
200             result = FSXA_SEMAPHORE_ERROR;
201             ERXA_LOG(result, 0, ("SEMAPHORE_ERROR: PLXAsem_gethandle failed"));
202          }
203       }
204       else if ( result != OK )
205       {
206          /* semaphore cration failed */
207          result = FSXA_SEMAPHORE_ERROR;
208          ERXA_LOG(result, 0, ("SEMAPHORE_ERROR: PLXAsemM_create failed"));
209       }
210       else
211       {
212          /* QAC */
213       }
```

SMELL Evaluation

	KLOC	Reported Deviations	False Positives	Limitations	Validated Deviations
CC1	3	32	2	4	26
CC2	19	72	20	24	28
CC6	15	16	0	3	13
CC4	14	107	14	13	80
CC5	15	9	0	3	6
	66	236	36	47	153

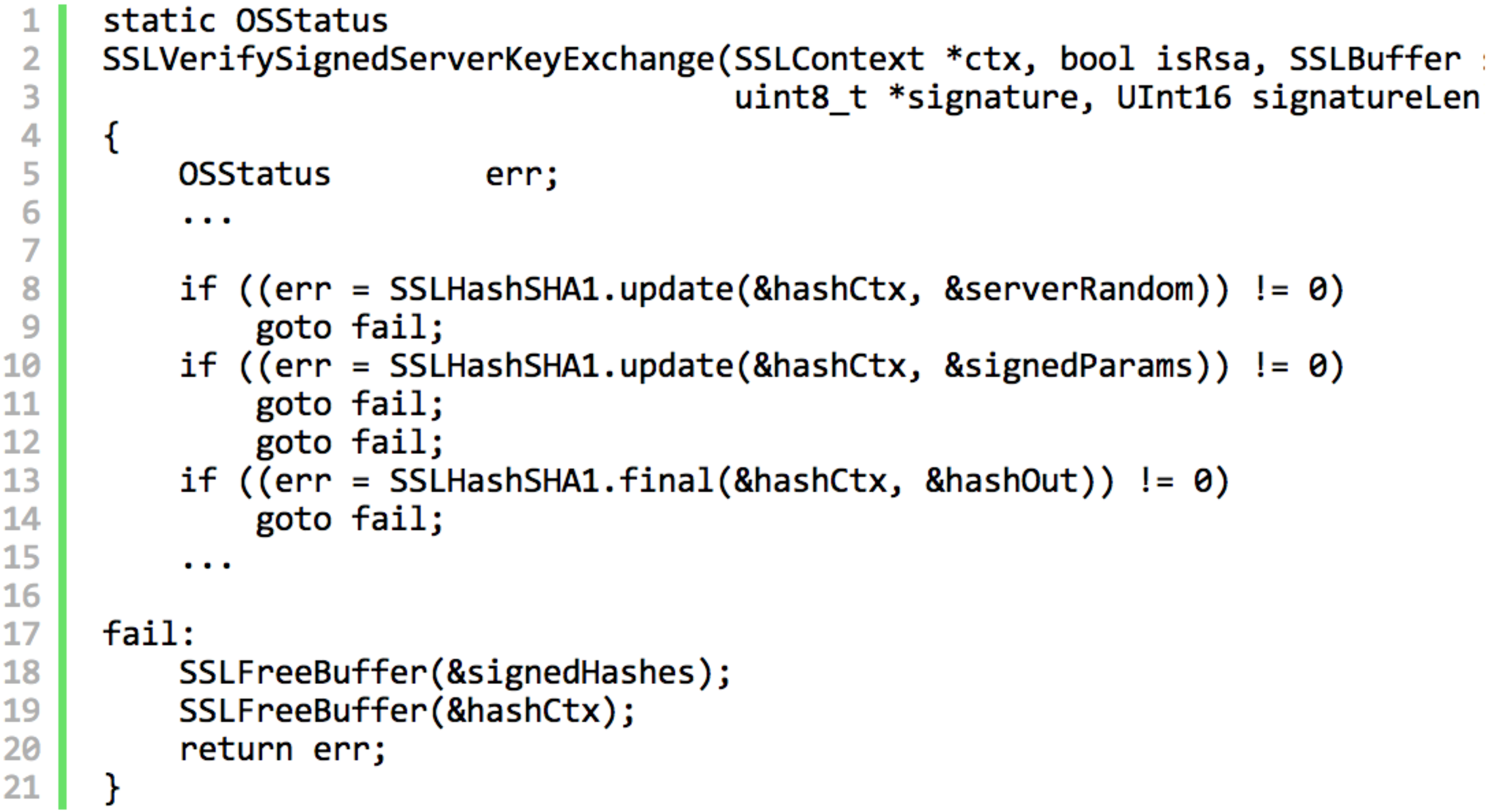
15%

20%

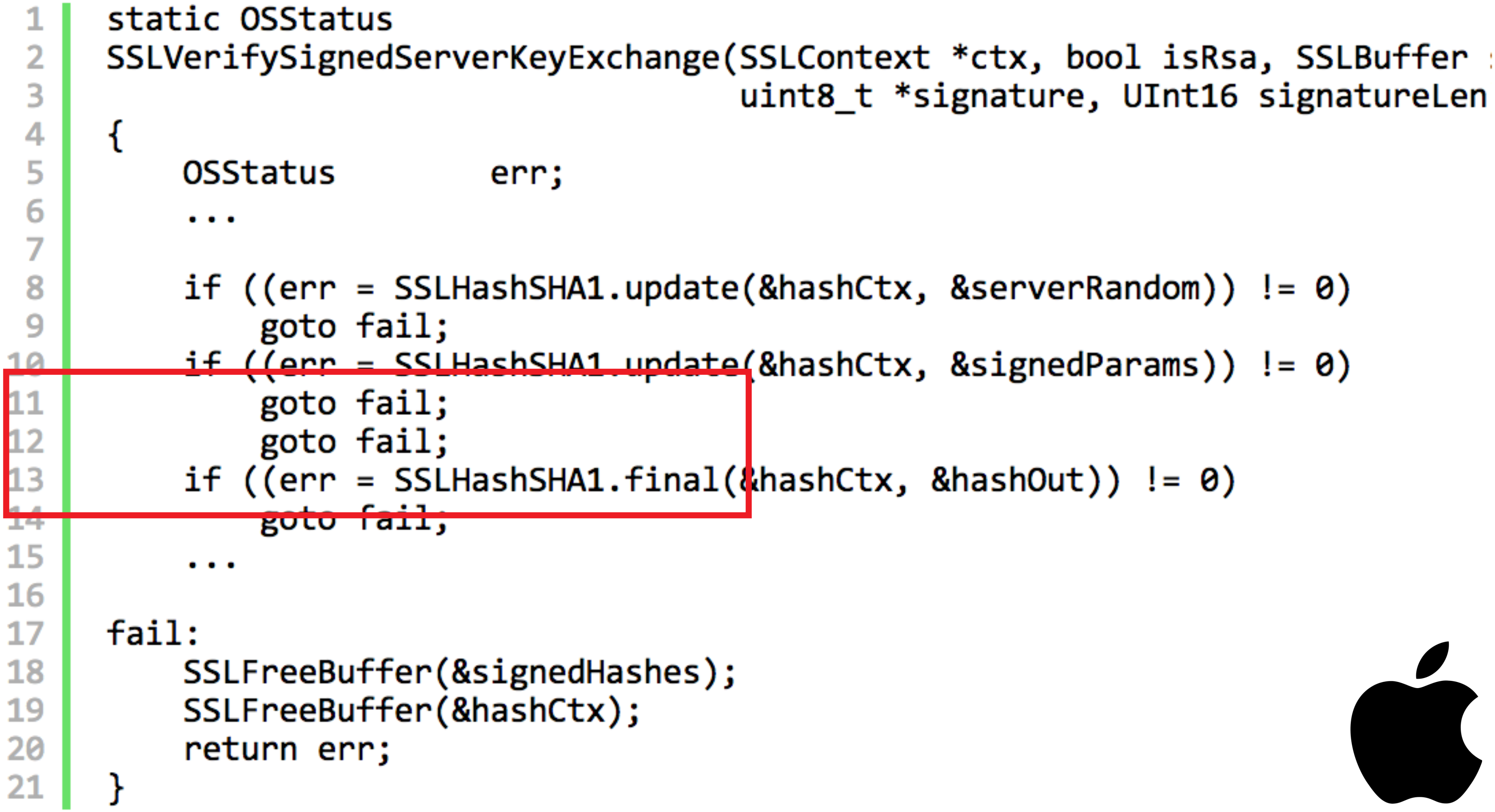
~2/Kloc

Error Handling In Embedded C

- Exception handling through return code idiom
- Almost 10% of code base
- 2 faults per KLOC in just error handling code
- Incorrect logging affects repair time = uptime
- Static analysis can help to spot problems



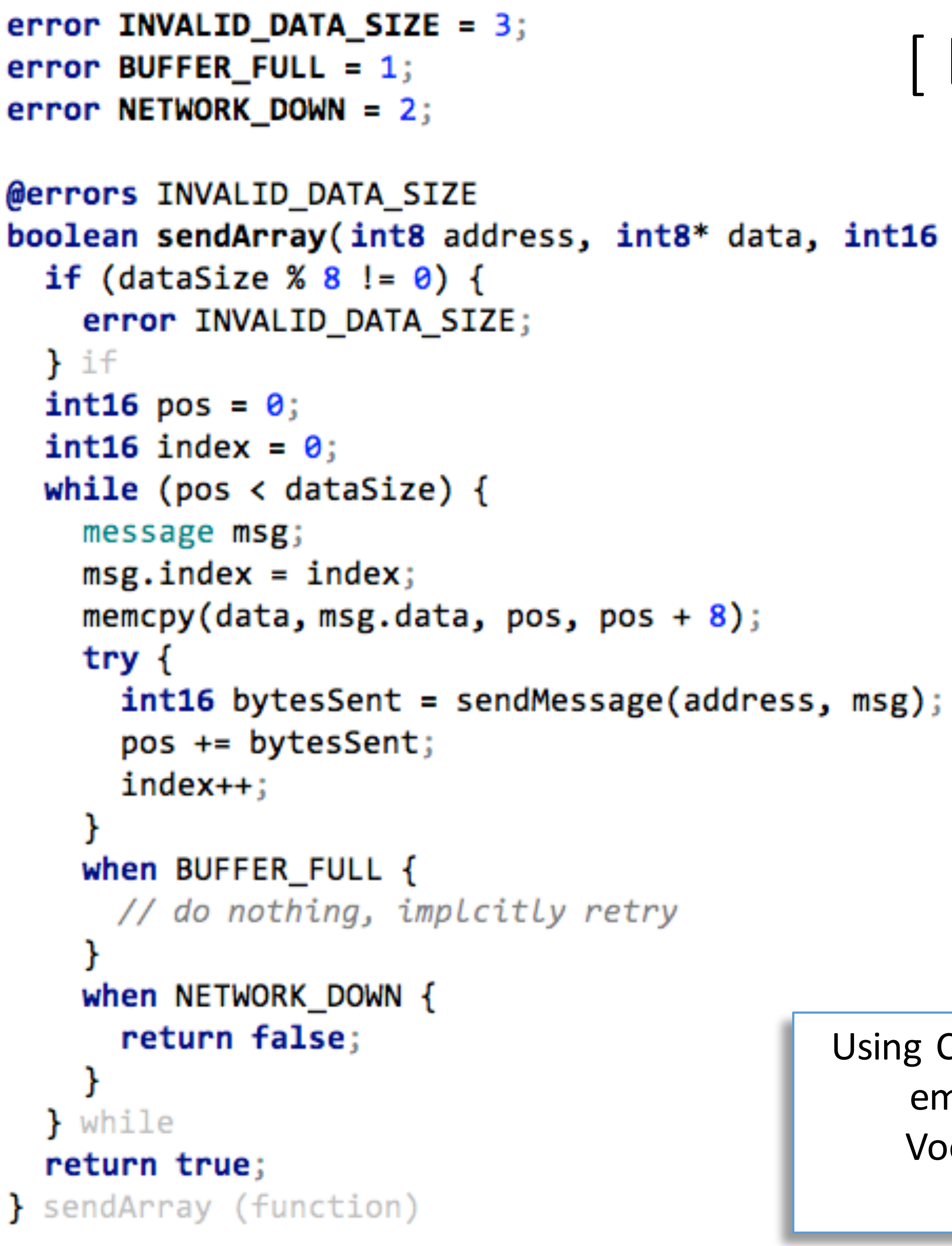
```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer
                                 uint8_t *signature, UInt16 signatureLen
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```



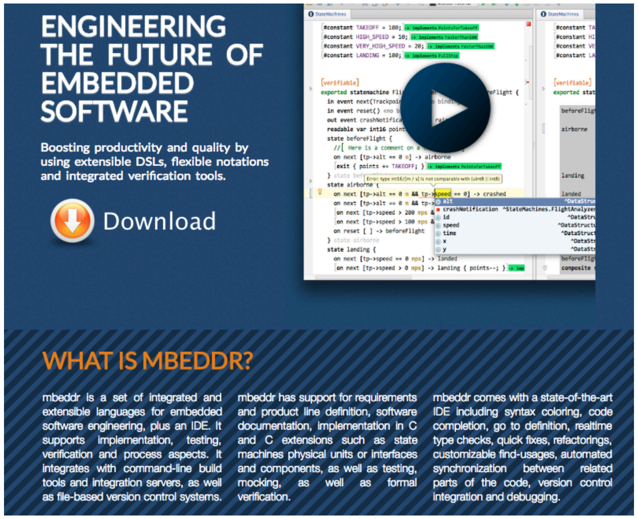
```
static OSStatus
SSLVerifySignedServerKeyExchange(SSLContext *ctx, bool isRsa, SSLBuffer
                                 uint8_t *signature, UInt16 signatureLen
{
    OSStatus        err;
    ...

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;
    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
        goto fail;
    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;
    ...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

2014: Security vulnerability in error handling of Apple’s Secure Socket Layer code
http://avandeursen.com/2014/02/22/gotofail-security/

[Language extensions to the rescue?]



```
error INVALID_DATA_SIZE = 3;
error BUFFER_FULL = 1;
error NETWORK_DOWN = 2;

@errors INVALID_DATA_SIZE
boolean sendArray(int8 address, int8* data, int16 dataSize) {
  if (dataSize % 8 != 0) {
    error INVALID_DATA_SIZE;
  } if
  int16 pos = 0;
  int16 index = 0;
  while (pos < dataSize) {
    message msg;
    msg.index = index;
    memcpy(data, msg.data, pos, pos + 8);
    try {
      int16 bytesSent = sendMessage(address, msg);
      pos += bytesSent;
      index++;
    }
    when BUFFER_FULL {
      // do nothing, implcitly retry
    }
    when NETWORK_DOWN {
      return false;
    }
  } while
  return true;
} sendArray (function)
```



ENGINEERING THE FUTURE OF EMBEDDED SOFTWARE

Boosting productivity and quality by using extensible DSLs, flexible notations and integrated verification tools.

WHAT IS MBEDDR?

mbeddr is a set of integrated and extensible languages for embedded software engineering, plus an IDE. It supports implementation, testing, verification and process aspects. It integrates with command-line build tools and integration servers, as well as file-based version control systems.

mbeddr has support for requirements and product line definition, software documentation, implementation in C and C extensions such as state machines physical units or interfaces and components, as well as testing, mocking, as well as formal verification.

mbeddr comes with a state-of-the-art IDE including syntax coloring, code completion, go to definition, realtime type checks, quick fixes, refactorings, customizable find-usages, automated synchronization between related parts of the code, version control integration and debugging.

Using C language extensions for developing embedded software: A case study
Voelter, Van Deursen, Kolb, Eberle.
OOPSLA 2015 .

Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao,
Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

This paper is included in the Proceedings of the
11th USENIX Symposium on
Operating Systems Design and Implementation.

October 6–8, 2014 • Broomfield, CO

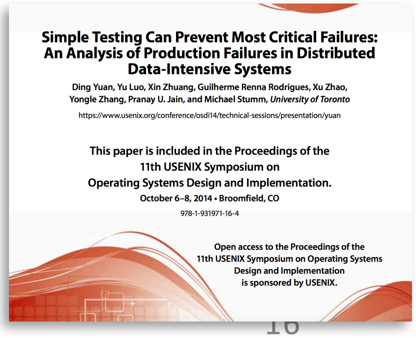
978-1-931971-16-4

Study Context

- 198 randomly sampled user-reported failures
- 5 data-intensive distributed systems
  - Cassandra, HBase, HDFS, MapReduce, Redis

- Widely used, all designed for high fault tolerance

- Studied all failure reports
- Manually reproduced 73 crashes.



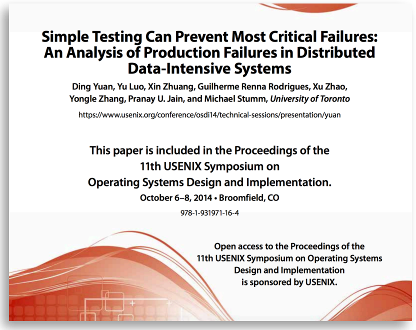
Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao,
Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

This paper is included in the Proceedings of the
11th USENIX Symposium on
Operating Systems Design and Implementation.
October 6–8, 2014 • Broomfield, CO
978-1-931971-16-4

Open access to the Proceedings of the
11th USENIX Symposium on Operating Systems
Design and Implementation
is sponsored by USENIX.



Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao,
Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

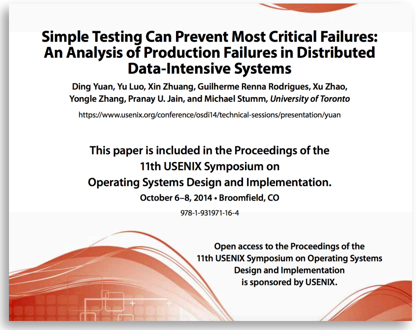
This paper is included in the Proceedings of the
11th USENIX Symposium on
Operating Systems Design and Implementation.
October 6–8, 2014 • Broomfield, CO

978-1-931971-16-4

Open access to the Proceedings of the
11th USENIX Symposium on Operating Systems
Design and Implementation
is sponsored by USENIX.

“Almost all (92%) of the catastrophic system failures are the result of incorrect handling of non-fatal errors explicitly signaled in software.”

Testing the Error Handler



Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

This paper is included in the Proceedings of the
11th USENIX Symposium on
Operating Systems Design and Implementation.
October 6–8, 2014 • Broomfield, CO
978-1-931971-16-4

Open access to the Proceedings of the
11th USENIX Symposium on Operating Systems
Design and Implementation
is sponsored by USENIX.

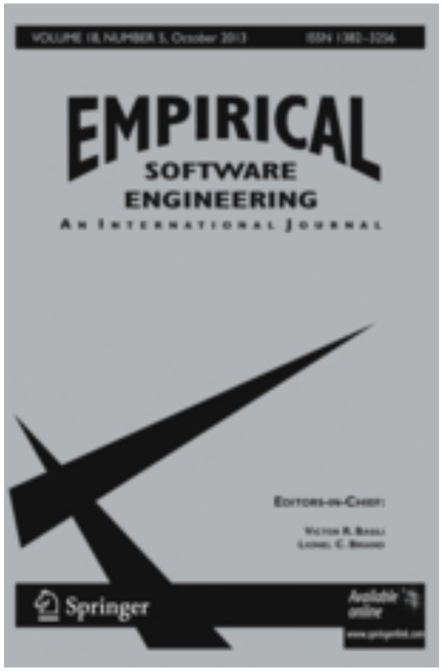
in 58% of the catastrophic failures,
the underlying faults
could easily have been detected
through simple testing of error handling code.

Causes:

i. the error handler is simply empty or only contains a log printing statement,

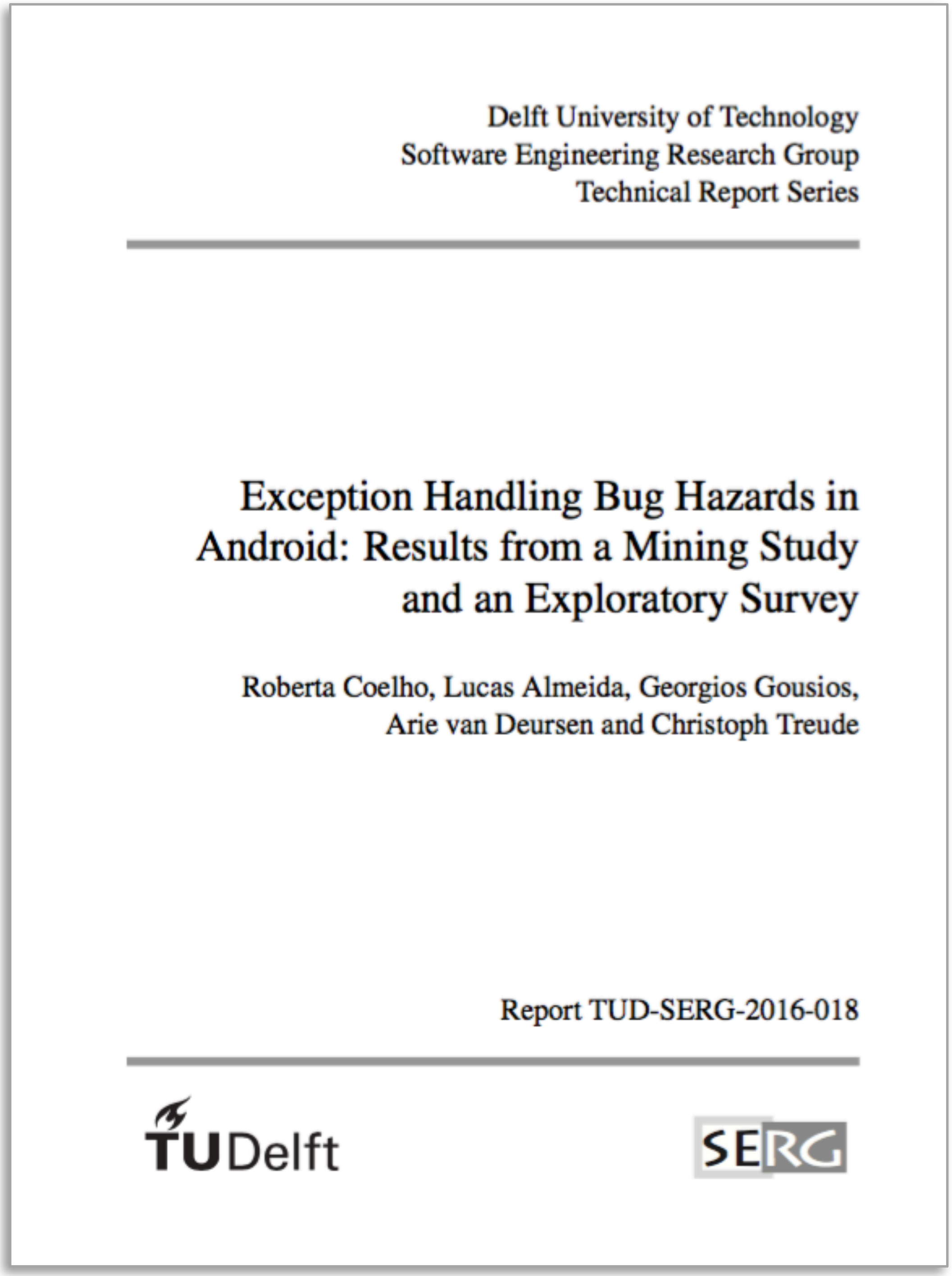
ii. the error handler aborts the cluster on an overly-general exception, and

iii. the error handler contains expressions like “FIXME” or “TODO” in the comments.



EMSE 2016

MSR 2015



Delft University of Technology
Software Engineering Research Group
Technical Report Series

Exception Handling Bug Hazards in Android: Results from a Mining Study and an Exploratory Survey

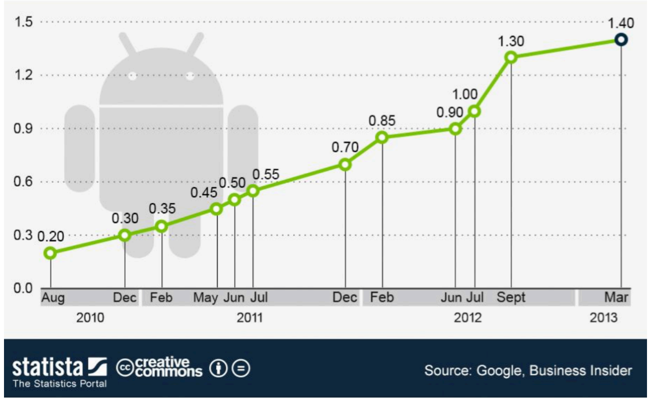
Roberta Coelho, Lucas Almeida, Georgios Gousios,
Arie van Deursen and Christoph Treude

Report TUD-SERG-2016-018

TUDelft SERG

Why Study Exceptions in Android?

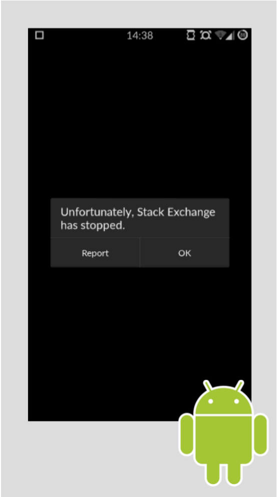
- Android is common
- App crashes are common



Source: Google, Business Insider

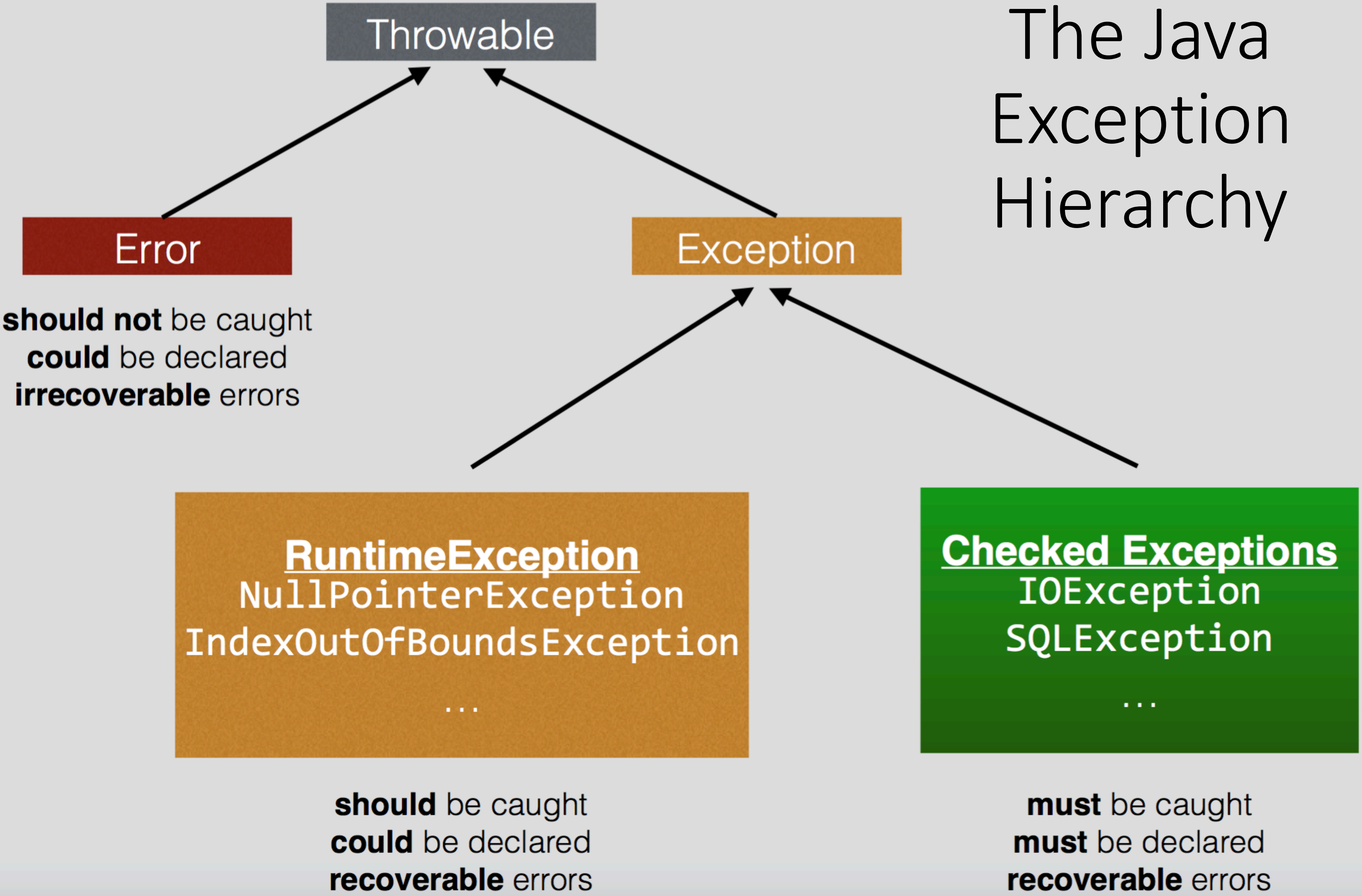
- Mix of platform, libraries, app

- For many apps source code available
- For many apps crash data available (issues)



Unfortunately, Stack Exchange has stopped.

The Java Exception Hierarchy



Throwable

Error

should not be caught
could be declared
irrecoverable errors

Exception

RuntimeException
NullPointerException
IndexOutOfBoundsException
. . .

should be caught
could be declared
recoverable errors

Checked Exceptions
IOException
SQLException
. . .

must be caught
must be declared
recoverable errors

Exception Propagation

Thrown exception

```
javax.servlet.ServletException: Something bad happened
    at com.example.myproject.OpenSessionInViewFilter.doFilter
    at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter
    at com.example.myproject.ExceptionHandlerFilter.doFilter
    ... 22 more
```

Wrappings

```
Caused by: com.example.myproject.MyProjectServletException
    at com.example.myproject.MyServlet.doPost
    at javax.servlet.http.HttpServlet.service
    at javax.servlet.http.HttpServlet.service
    at org.mortbay.jetty.servlet.ServletHolder.handle
    at org.mortbay.jetty.servlet.ServletHandler$CachedChain.doFilter
    at com.example.myproject.OpenSessionInViewFilter.doFilter
    ... 27 more
Caused by: org.hibernate.exception.ConstraintViolationException: could not
insert: [com.example.myproject.MyEntity]
    at org.hibernate.exception.SQLStateConverter.convert
    at org.hibernate.exception.JDBCExceptionHelper.convert
    at org.hibernate.id.insert.AbstractSelectingDelegate.performInsert
    at org.hibernate.persister.entity.AbstractEntityPersister.insert
    ... 32 more
```

Root cause

```
Caused by: java.sql.SQLException: Violation of unique constraint
MY_ENTITY_UK_1: duplicate value(s) for column(s) MY_COLUMN in statement
[...]
    at org.hsqldb.jdbc.Util.throwError
    at org.hsqldb.jdbc.jdbcPreparedStatement.executeUpdate
    at com.mchange.v2.c3p0.impl.NewProxyPreparedStatement.executeUpdate
    at org.hibernate.id.insert.AbstractSelectingDelegate.performInsert
    ... 54 more
```

Exception Handling “Bug Hazards”

1. Can the information available in exception stack traces reveal exception handling bug hazards in both the Android applications and framework?

a. Can the root exceptions reveal bug hazards?

b. Can the exception types reveal bug hazards?

c. Can the exception wrappings reveal bug hazards?

Data extraction

Search GHTorrent and Google Code for `android`

2,542 repositories
589 with stack traces
482 after manual filtering

31,592 issues
4,042 with stack traces

788 repositories
183 with stack traces
157 after manual filtering

127,456 issues
1,963 with stack traces

6,005 issues with stack traces from **539** projects

Data Processing

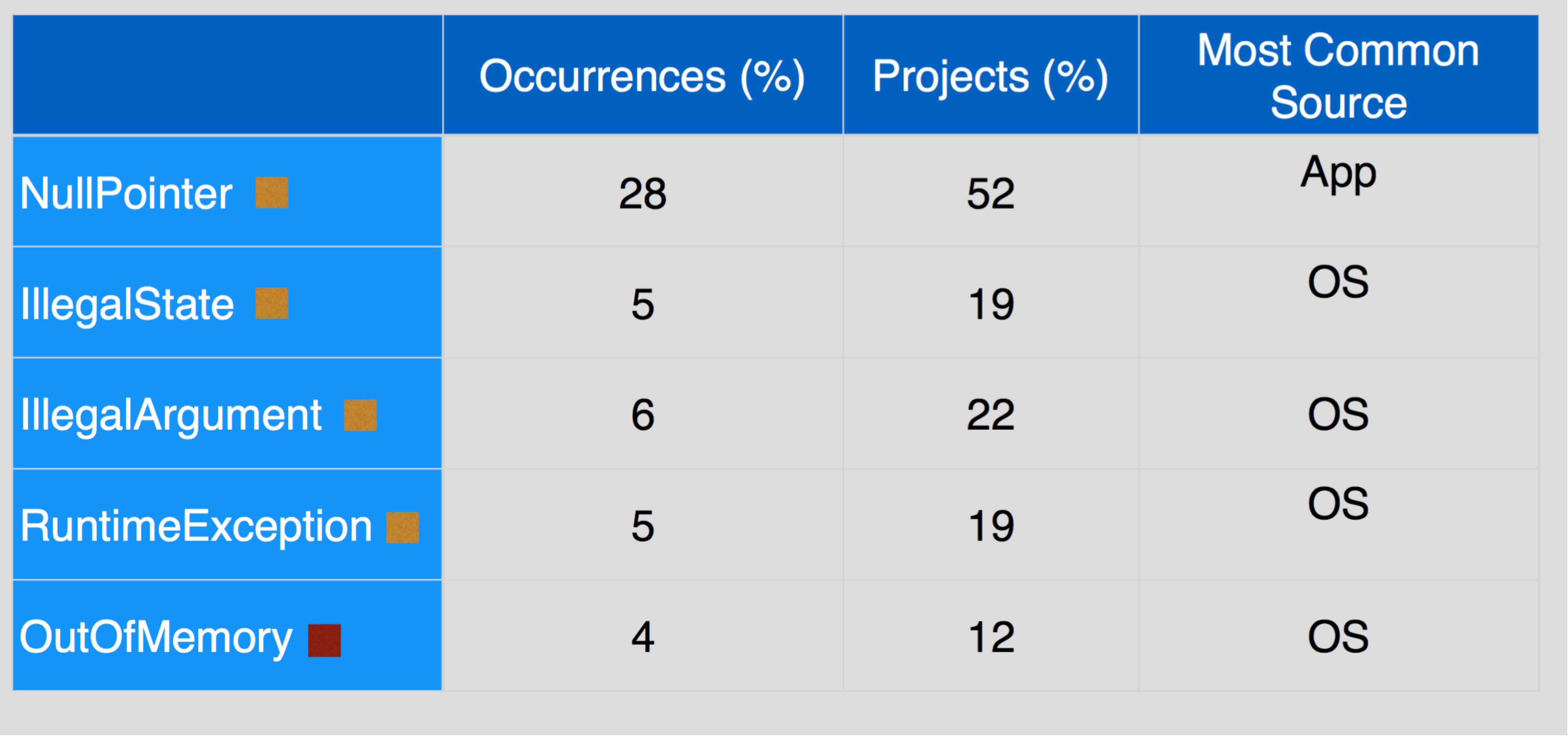
539 projects:

- Identify external libraries (static byte code analysis)
- Identify custom (checked) exception types (source code analysis)

6005 exceptions

- Extract stack trace using heuristics
- Check javadocs for thrown non-checked exceptions

Most Common Exceptions



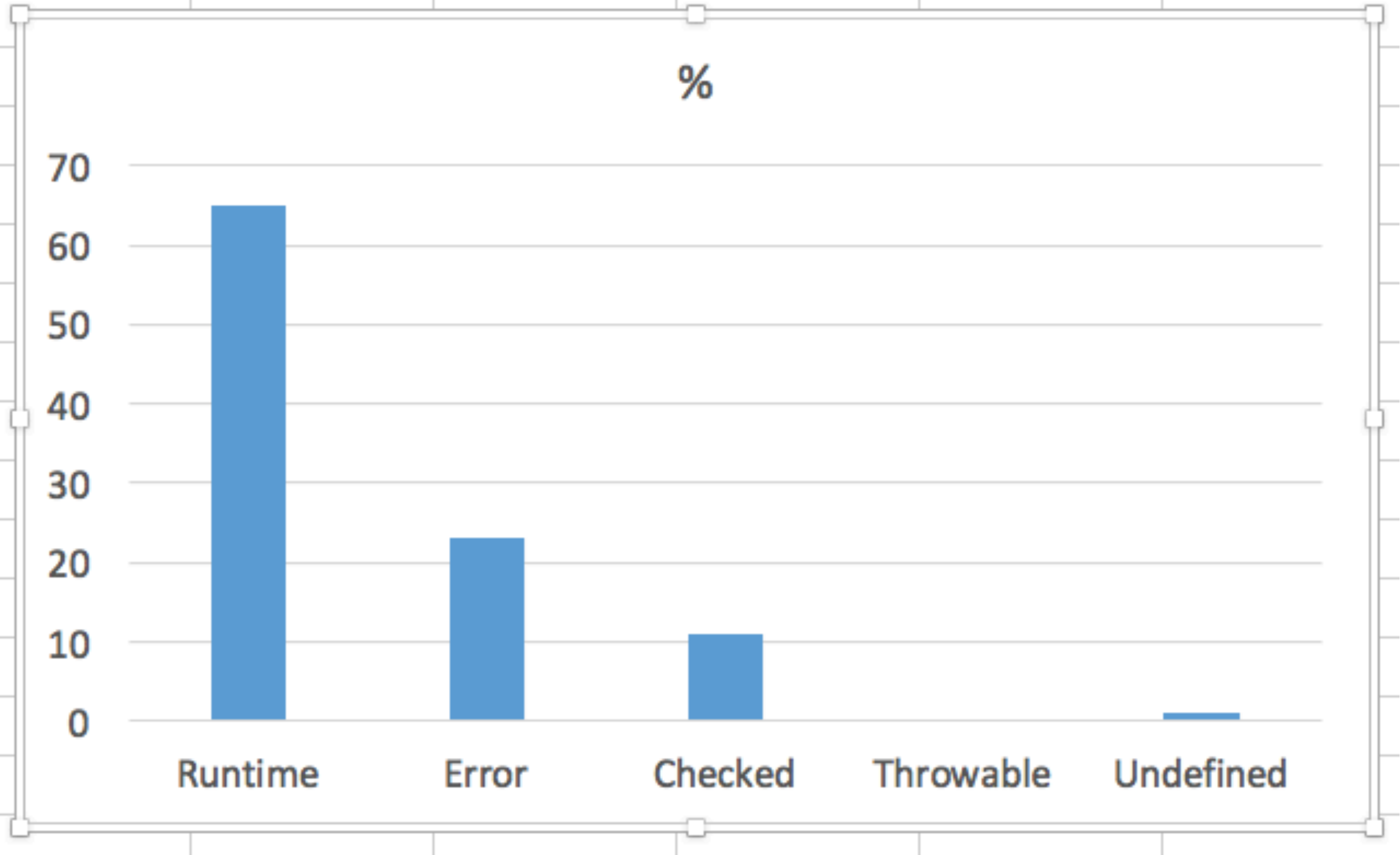
	Occurrences (%)	Projects (%)	Most Common Source
NullPointer 🟧	28	52	App
IllegalState 🟧	5	19	OS
IllegalArgument 🟧	6	22	OS
RuntimeException 🟧	5	19	OS
OutOfMemory 🟥	4	12	OS

Most faulty areas

	Occurrences (%)
Program logic	52
Resource handling	23
Security	4
Concurrency	3
Backward compatibility	4

Programming logic and resource handling account for 75% of all exceptions

Most Common Types

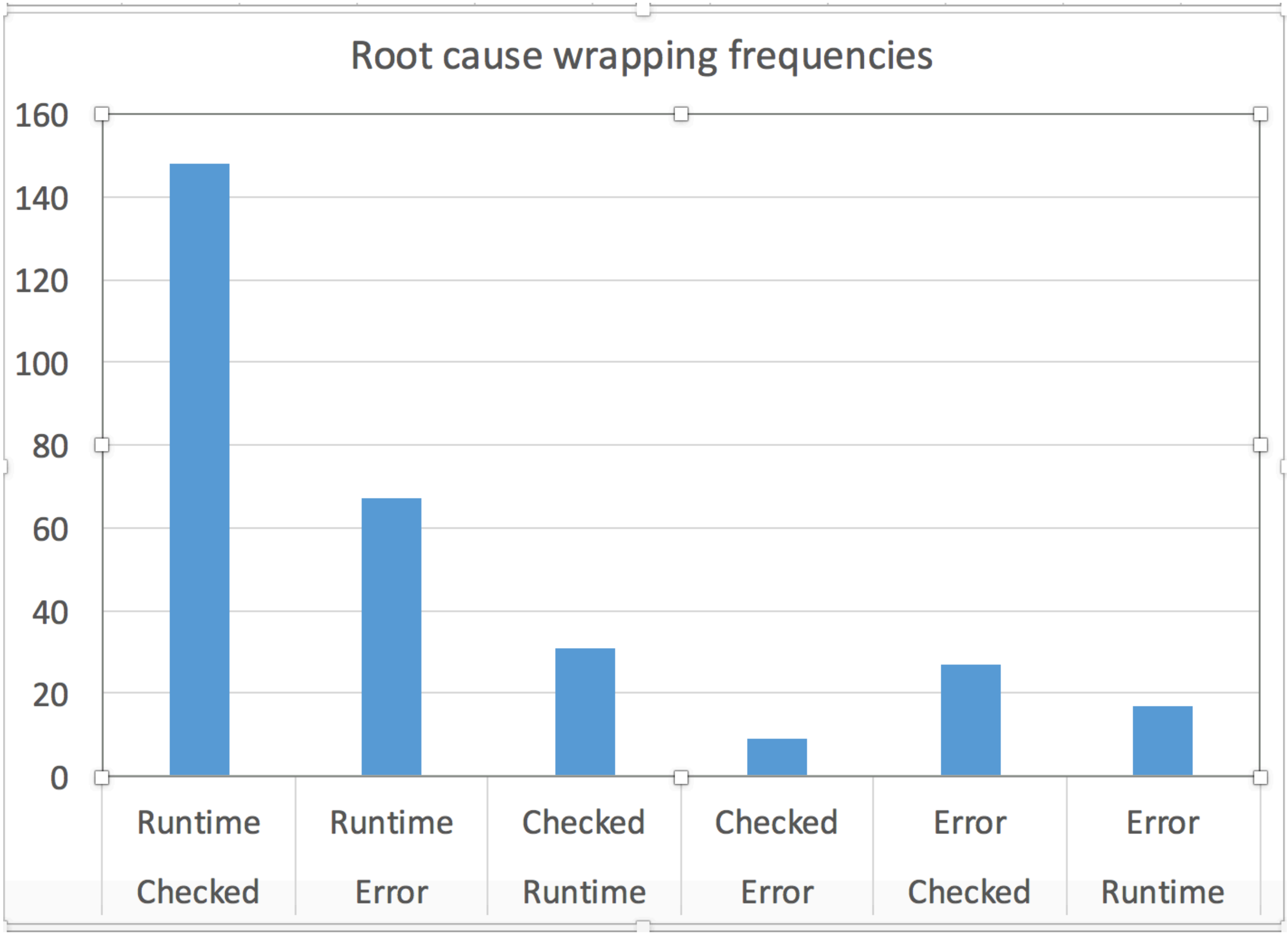


Exception Interfaces

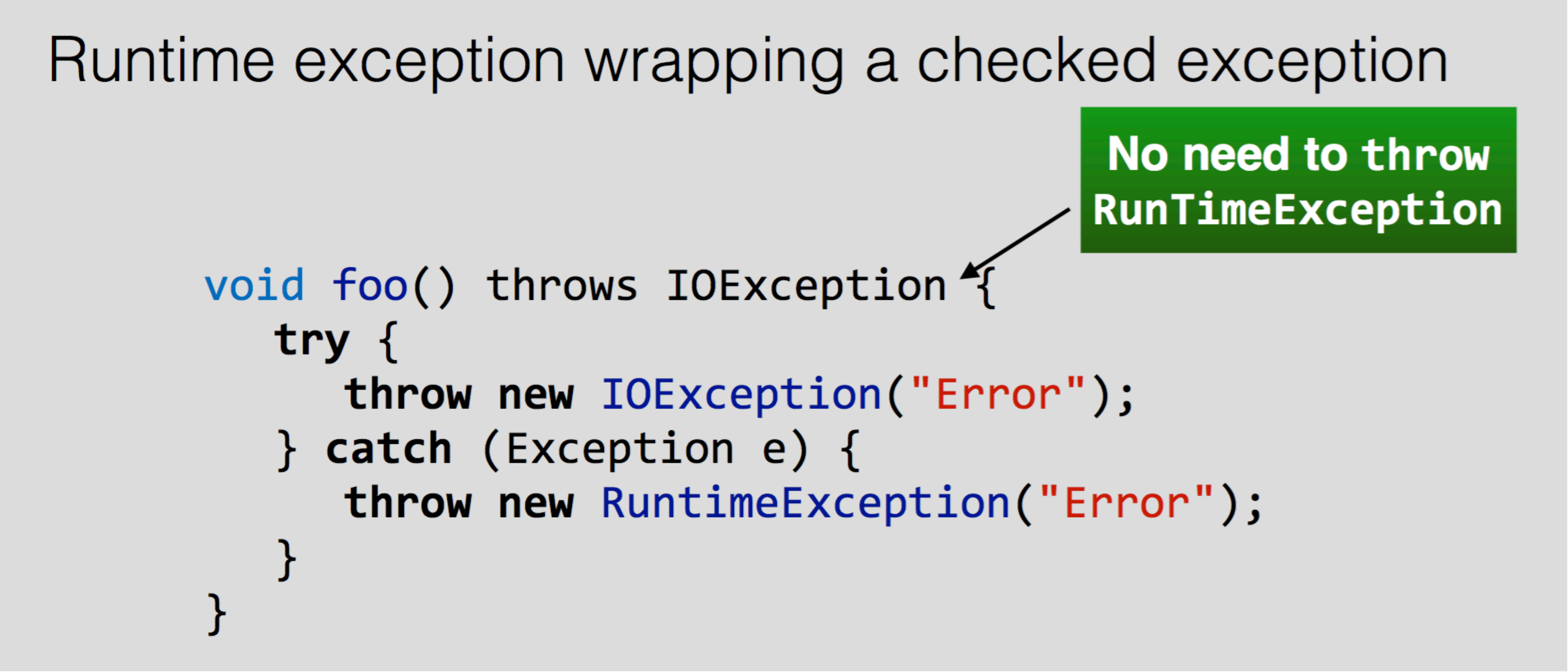
- 65% Runtime Exceptions?
- 4% programmatically thrown (79 cases)
- *Not* documented via “throws” signature (except 1 case)

- Surprise finding: *checked* exceptions can go undocumented too.
  - (Gingerbread JNI method could throw checked exception that was not documented)

Cross Type Wrappings



The 50% case:
Checked to Runtime

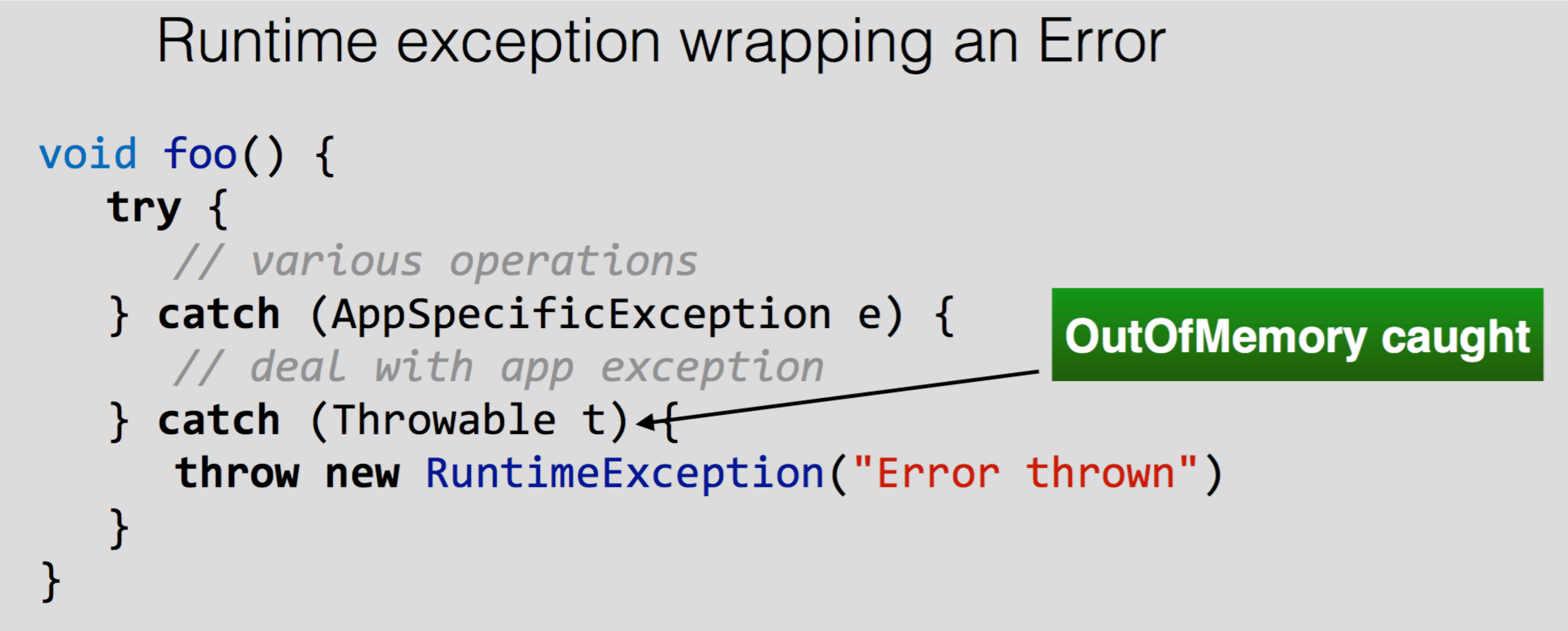


Runtime exception wrapping a checked exception

No need to throw RunTimeException

```
void foo() throws IOException {
    try {
        throw new IOException("Error");
    } catch (Exception e) {
        throw new RuntimeException("Error");
    }
}
```

The 22% case:
Error to Runtime (dangerous)

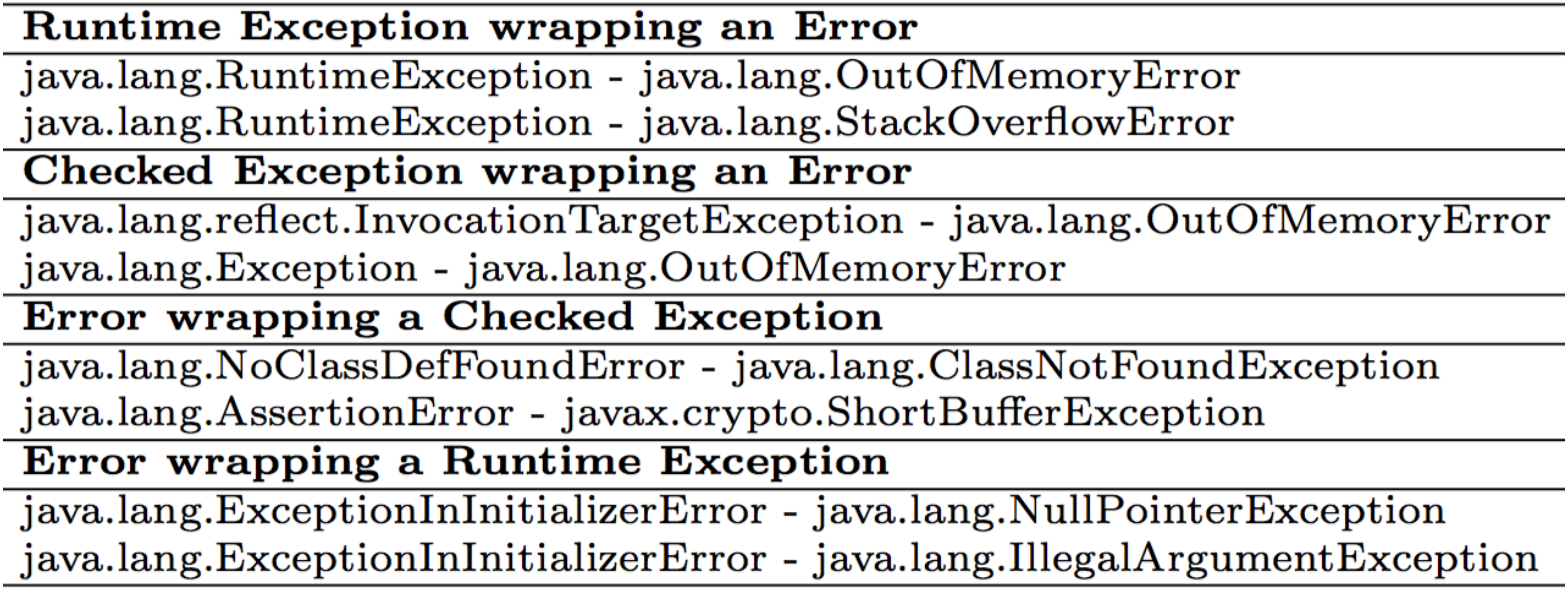


Runtime exception wrapping an Error

```
void foo() {
   try {
      // various operations
   } catch (AppSpecificException e) {
      // deal with app exception
   } catch (Throwable t) {
      throw new RuntimeException("Error thrown")
   }
}
```

OutOfMemory caught

Other Error-Related Wrappings

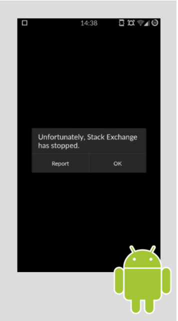


Runtime Exception wrapping an Error
java.lang.RuntimeException - java.lang.OutOfMemoryError
java.lang.RuntimeException - java.lang.StackOverflowError
Checked Exception wrapping an Error
java.lang.reflect.InvocationTargetException - java.lang.OutOfMemoryError
java.lang.Exception - java.lang.OutOfMemoryError
Error wrapping a Checked Exception
java.lang.NoClassDefFoundError - java.lang.ClassNotFoundException
java.lang.AssertionError - javax.crypto.ShortBufferException
Error wrapping a Runtime Exception
java.lang.ExceptionInInitializerError - java.lang.NullPointerException
java.lang.ExceptionInInitializerError - java.lang.IllegalArgumentException

Android Exception Handling: Findings

- Programming mistakes common cause:
  - 50% of reported stack traces
  - Null pointer exceptions most prominent (27%)
- Java Errors are wrapped in (checked) exceptions
  - E.g.: OutOfMemory wrapped in checked exception
- Thrown RuntimeExceptions are not documented
  - Occur in 4.4% of traces
- Undocumented checked exceptions occur:
  - raised by native C code, *not declared* in JNI interface

Do Android Developers Agree?



Unfortunately, Stack Exchange has stopped.

a. How do Android devs deal with exceptions?

b. How do NullPointerExceptions impact Android development?

c. How do cross-type wrappings impact Android development?

d. Are developers aware of JNI’s undocumented checked exceptions? [Answer: *no*]

e. How do developers prevent apps from crashing?

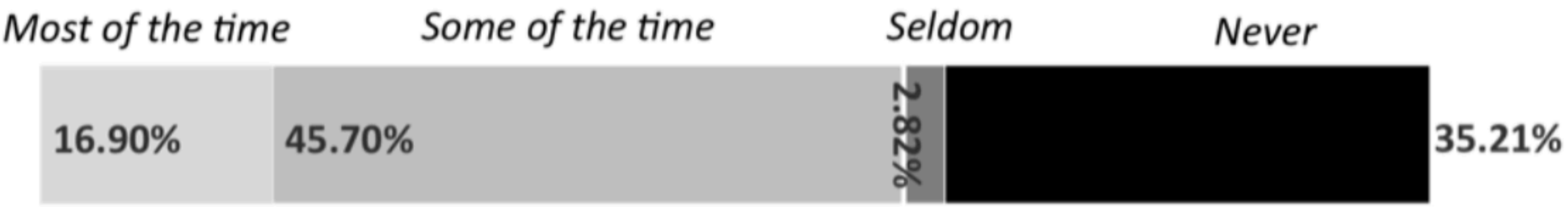
Experimental Setup

- Questionnaire to devs of Android apps under study
  - General exception handling policies
  - Bug hazards identified in our study
- 13 open questions, 5 Likert Scale questions, 10 multiple choice questions

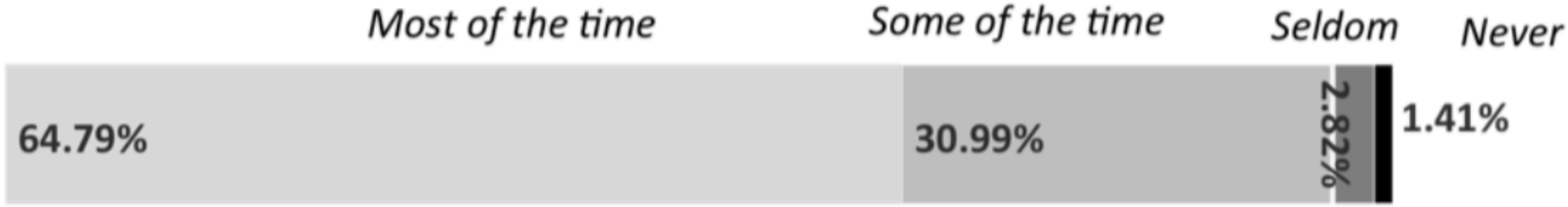
- Emailed 1824 devs; received 71 valid responses
- 85% over 2 years of Java and Android experience

- Open answers coded by two researchers

Dealing with Exceptions

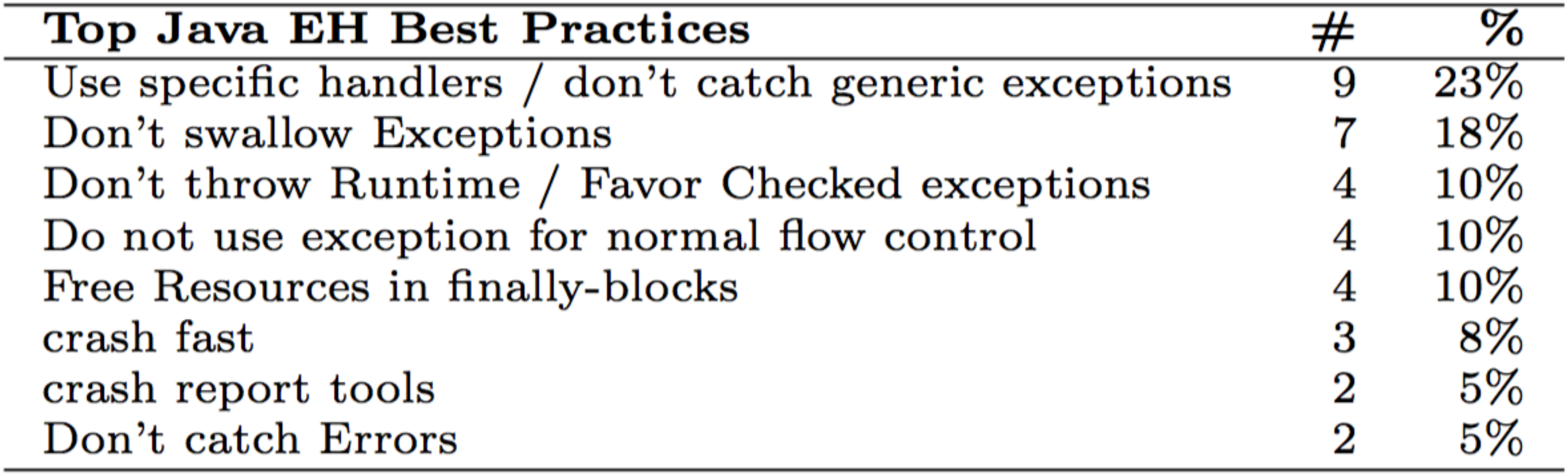


Need to throw an exception



Need to handle an exception

Exception Handling Best Practices

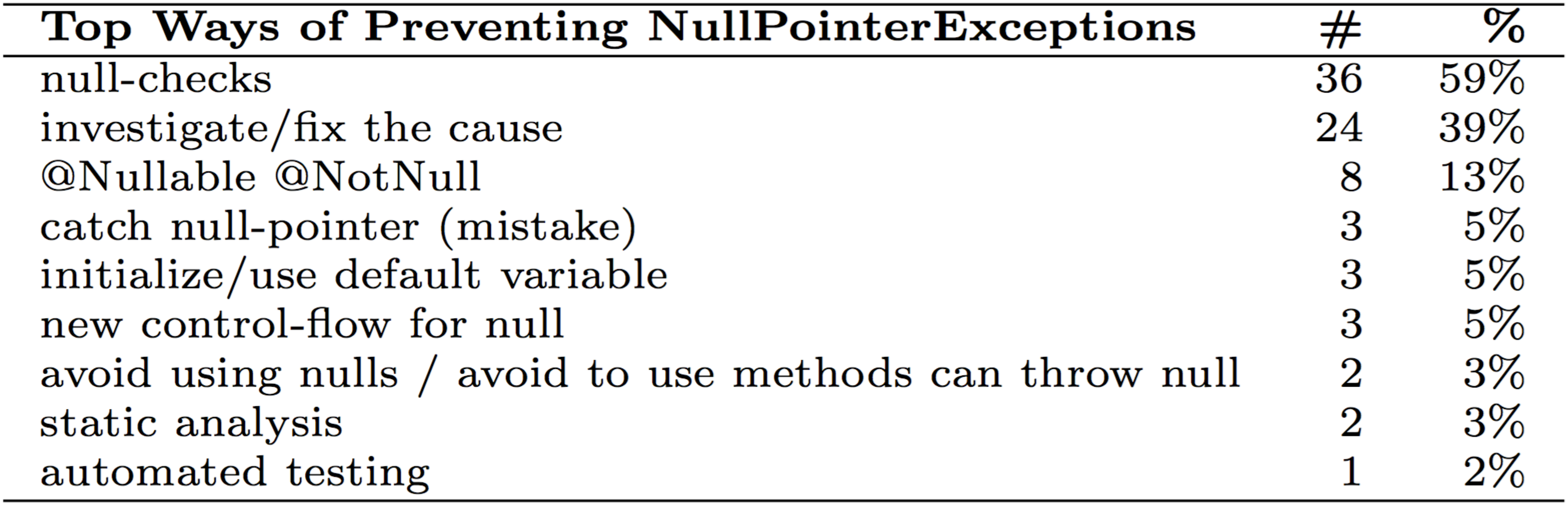


Top Java EH Best Practices	#	%
Use specific handlers / don't catch generic exceptions	9	23%
Don't swallow Exceptions	7	18%
Don't throw Runtime / Favor Checked exceptions	4	10%
Do not use exception for normal flow control	4	10%
Free Resources in finally-blocks	4	10%
crash fast	3	8%
crash report tools	2	5%
Don't catch Errors	2	5%

“Android destroys and recreates itself all of the time
(especially during screen rotation).
If you do not handle that it will crash on you every time.

With the complexity of
an activity with a fragment that has fragments and
each of those fragments has custom objects and variables
that need to be retained (so saved and put back)
or recreated such as views
it can get complex
if you don’t have an understanding of
how the Android life cycle works.” [D43].

2b. Dealing with Null Pointers



Top Ways of Preventing NullPointerExceptions	#	%
null-checks	36	59%
investigate/fix the cause	24	39%
@Nullable @NotNull	8	13%
catch null-pointer (mistake)	3	5%
initialize/use default variable	3	5%
new control-flow for null	3	5%
avoid using nulls / avoid to use methods can throw null	2	3%
static analysis	2	3%
automated testing	1	2%

“NullPointerExceptions can happen
pretty much anywhere.
The Android Fragment system comes to mind.

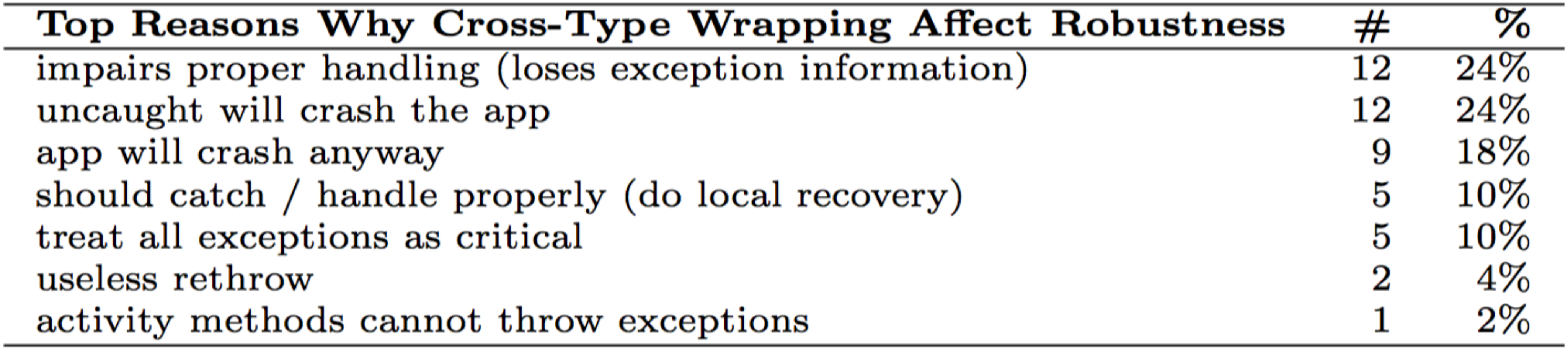
Often, it is possible to find yourself in a state
where getActivity() is null within the Fragment
during certain points in the life cycle,
and that is something I have to plan for.”
[D56]

Determine why the object was null and attempt to fix this situation.

In the case of external API calls which return null, then check for null (the quick-and-dirty way).

For internal calls, use @NotNull and @Nullable annotations to provide more guidance on when an object "may be" and "should never be" null." [D42]

2c. Dealing with Wrappings



Top Reasons Why Cross-Type Wrapping Affect Robustness	**#**	**%**
impairs proper handling (loses exception information)	12	24%
uncaught will crash the app	12	24%
app will crash anyway	9	18%
should catch / handle properly (do local recovery)	5	10%
treat all exceptions as critical	5	10%
useless rethrow	2	4%
activity methods cannot throw exceptions	1	2%

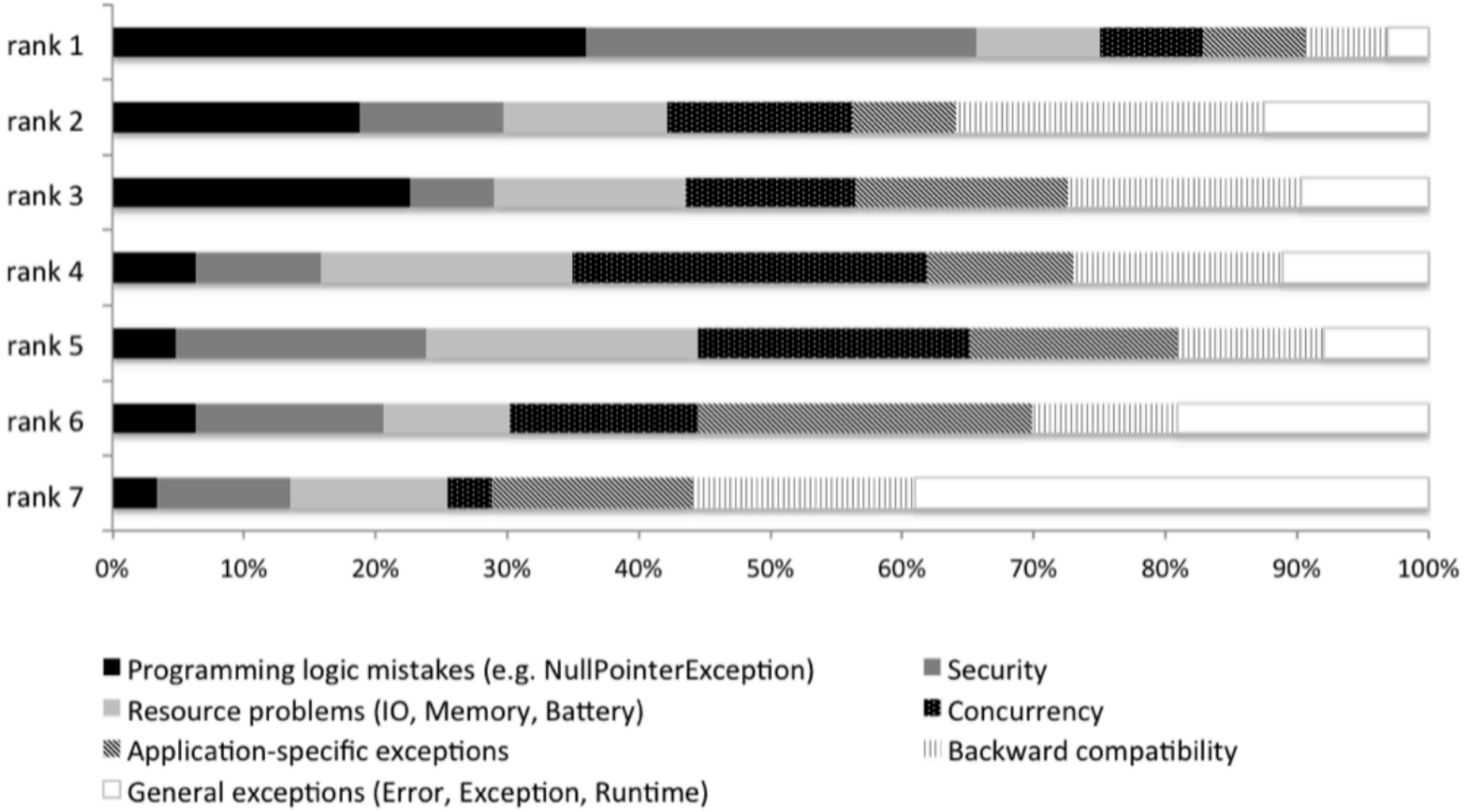
“Honestly, there are a lot of very unskilled Java programmers out there writing Android apps.

When they encounter NPE,
they tend to null-check that variable,
which just puts a bandage on the problem
and causes other failures (usually also NPE’s)
later on in the application’s lifecycle.” [D42]

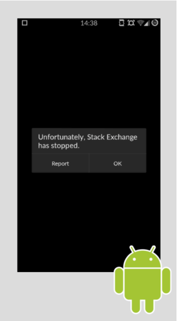
The Android framework is what adds the complexity in figuring out what caused an exception.

Because more often than not, the error is triggered from the framework as a result of something else you did.” [D58];

Fault Sensitive Areas



Exception Handling in Android



Unfortunately, Stack Exchange has stopped.

- Developers are struggling with Java’s exceptions
- Checked exceptions give a false sense of safety
  - Undocumented runtime exceptions are common too

- Programming errors are a main cause of exception-based crashes

- Exception handling is an architectural concern:
  - Needs to be consistent across components and libraries
  - Chains of inconsistent wrappings hamper understandability

Logness

Mining Log Data for Relevant Exceptions

Joint work with Peter Evers (TU Delft)
Maurício Aniche (TU Delft)
Maikel Lobbezoo (Adyen)

adyen

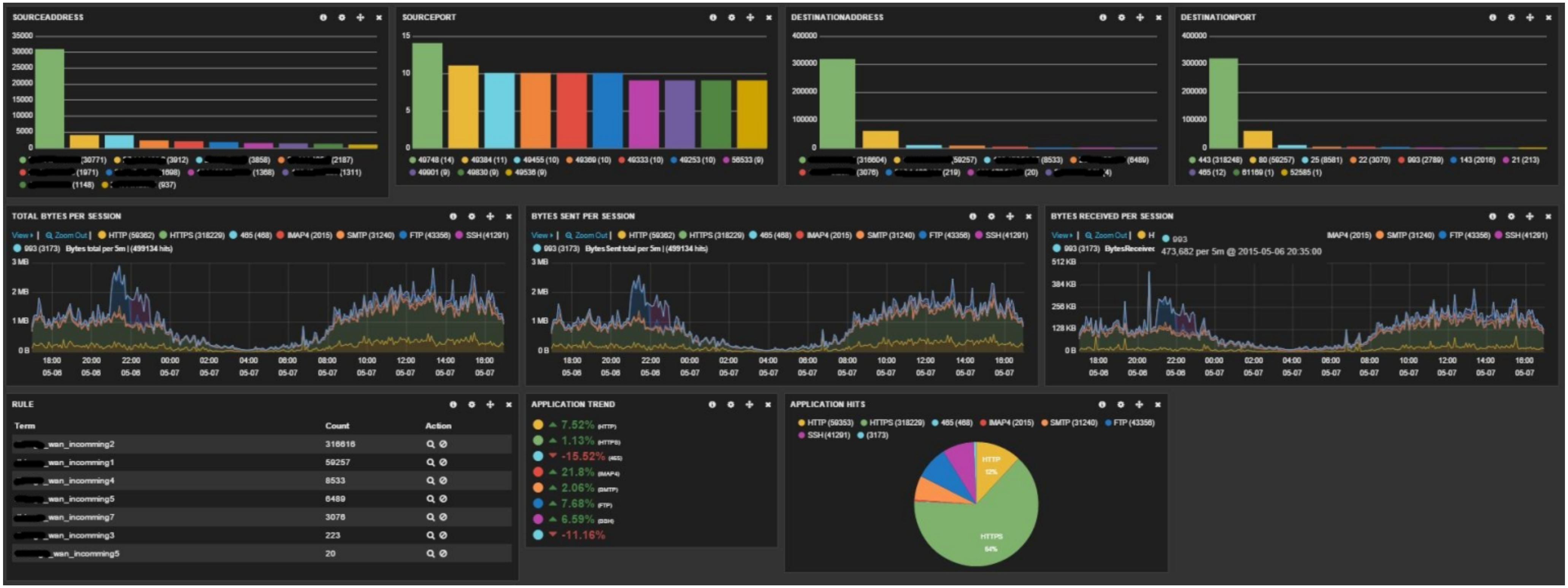
- Founded in 2006
- 6th Unicorn Startup in Europe (over $1 billion venture capital)
- Payment Service Provider
- Enables merchants to accept payments from anywhere in the world
- 450 people in 10 countries

Logging at Internet Payment Scale

- Around 500 million log messages per day
- Around 500,000 warnings / errors per day
- > 100 GB log data per day
- Sensitive data logged in encrypted form

Monitoring

- Weekly release cycle
- Different monitors and custom alerts in place



RULE

Term	Count	Action
[redacted]_wan_incomming2	316616	🔍 ⊘
[redacted]_wan_incomming1	59257	🔍 ⊘
[redacted]_wan_incomming4	8533	🔍 ⊘
[redacted]_wan_incomming5	6489	🔍 ⊘
[redacted]_wan_incomming7	3076	🔍 ⊘
[redacted]_wan_incomming3	223	🔍 ⊘
[redacted]_wan_incomming5	20	🔍 ⊘

Log Data Analytics: The ELK stack

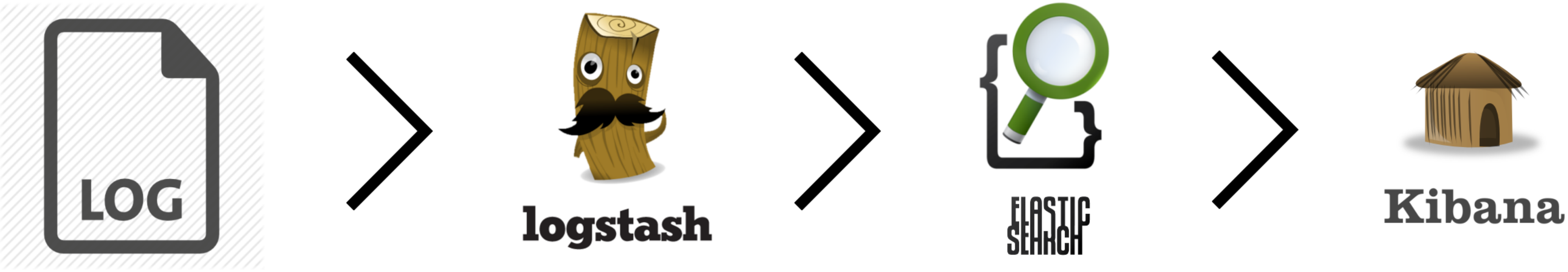
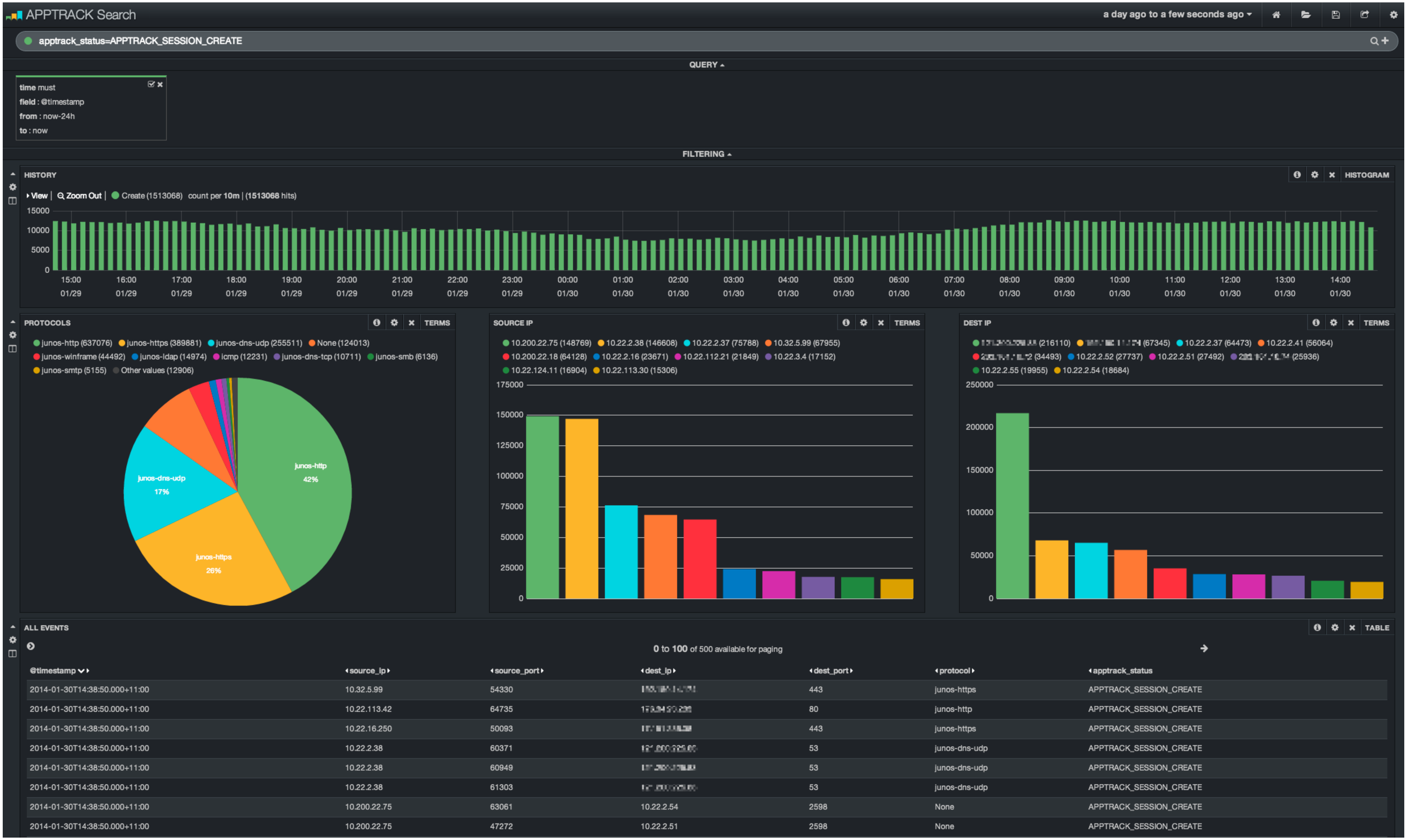


Image credit: www.neteye-blog.com



apptrack_status=APPTRACK_SESSION_CREATE

time must
field : @timestamp
from : now-24h
to : now

0 to 100 of 500 available for paging

@timestamp	source_ip	source_port	dest_ip	dest_port	protocol	apptrack_status
2014-01-30T14:38:50.000+11:00	10.32.5.99	54330	[redacted]	443	junos-https	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.22.113.42	64735	[redacted]	80	junos-http	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.22.16.250	50093	[redacted]	443	junos-https	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.22.2.38	60371	[redacted]	53	junos-dns-udp	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.22.2.38	60949	[redacted]	53	junos-dns-udp	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.22.2.38	61303	[redacted]	53	junos-dns-udp	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.200.22.75	63061	10.22.2.54	2598	None	APPTRACK_SESSION_CREATE
2014-01-30T14:38:50.000+11:00	10.200.22.75	47272	10.22.2.51	2598	None	APPTRACK_SESSION_CREATE

http://ifconfig-a.com/?p=8

In practice: "tail –f", grep

```
2009-09-29 12:53:32,644 error [com.adyen.rpl.payment.openinvoice.OpenInvoiceCaller] (TP-Processor114) org.codehaus.xfire.XFireRuntimeException: Could not invoke service.. Nested exception is org.codehaus.xfire.fault.XFireFault: internal
Could not send the message: method name=retrieveDetail, method params[0=com.adyen.services.openinvoice.OpenInvoiceDetailRequest], data params[0=com.adyen.services.openinvoice.OpenInvoiceDetailRequest]
2009-09-29 12:53:32,644 error [com.adyen.actions.hpp.CompleteOpenInvoice] (TP-Processor114) Problem handling payment, service exception: Could not invoke service.. Nested exception is org.codehaus.xfire.fault.XFireFault: internal Could n
ot send the message: method name=retrieveDetail, method params[0=com.adyen.services.openinvoice.OpenInvoiceDetailRequest], data params[0=com.adyen.services.openinvoice.OpenInvoiceDetailRequest]
2009-09-29 12:53:33,040 INFO  [com.adyen.actions.hpp.SelectPaymentMethod] (TP-Processor34) Received request from 213.133.51.5: blockedMethods= & orderData= & currencyCode=EUR & shopperReference= & countryCode= & skinCode=LxC3kOrf & merch
antReference=[illegible] & shopperEmail=[illegible] & merchantAccount=[illegible] & shipBeforeDate=2009-10-06 & paymentAmount=6900 & skipSelection=true & shopperLocale=NL & sessionValidity=2009-1
0-04T12:42:53+02:00 & allowedMethods= & merchantSig=[illegible] & recurringContract= &
2009-09-29 12:53:33,040 INFO  [com.adyen.actions.hpp.SelectPaymentMethod] (TP-Processor34) Setting cookie: Cookie
        [ name      : hppsession
          value     : [illegible]
          domain    : null
          maxage    : 1209600
          path      : /hpp/
          secure    : true ]

2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - bankTransfer_DE) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - ideal) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - bankTransfer_BE) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - mc) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - bankTransfer_NL) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - visa) for currencies ([EUR]) with constraints : []
2009-09-29 12:53:33,041 INFO  [com.adyen.actions.hpp.util.PaymentMethodUtil] (TP-Processor34) Brand applicable: MerchantPaymentMethod ([illegible] - elv) for currencies ([EUR]) with constraints : [elv (null) (EUR) provided by acq
[illegible]
```

The Problem

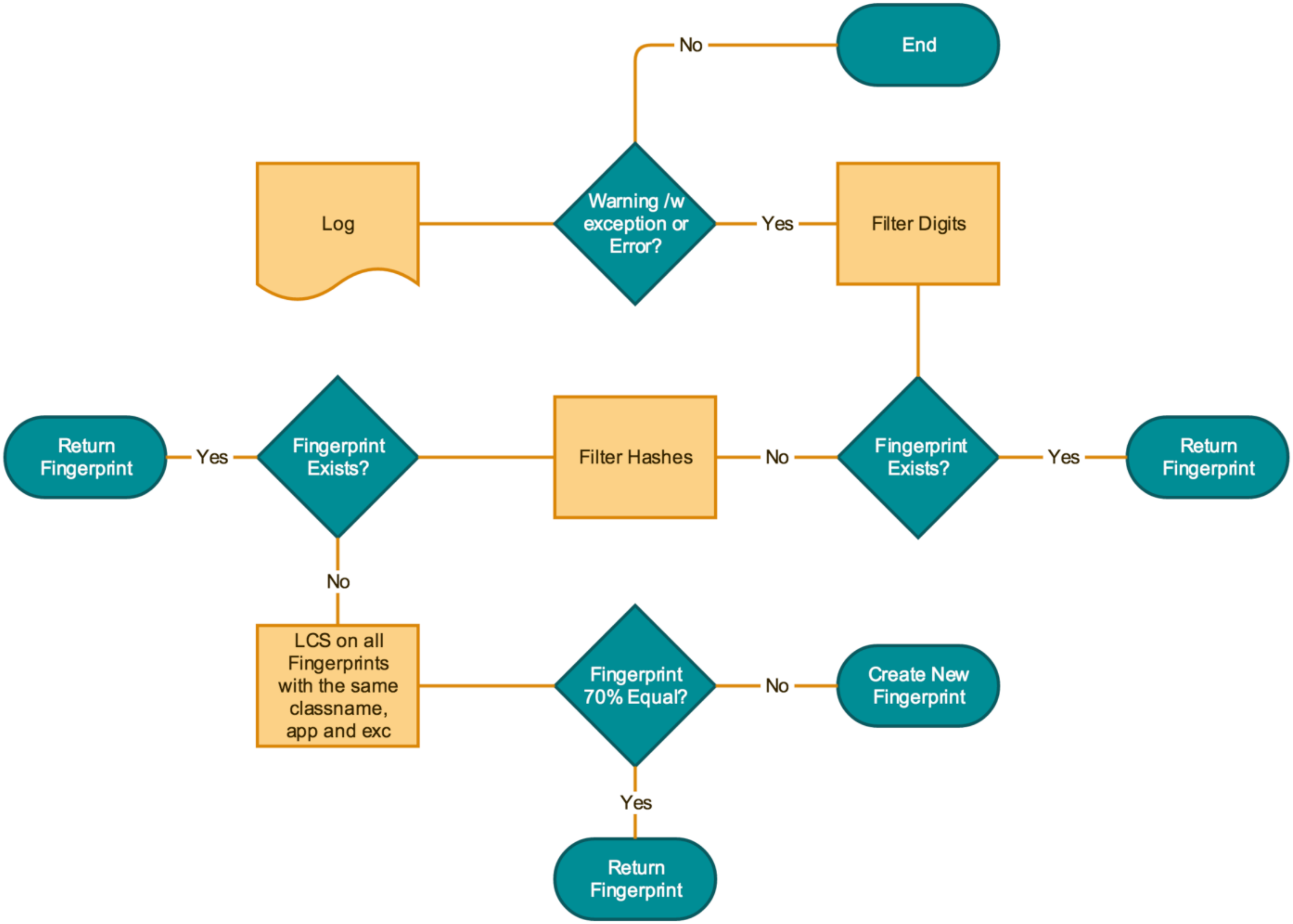
- Small problems are easily missed
- Monitoring is done on one server at the time
- Requires domain knowledge about (almost) everything
- Hard to distinguish ‘normal’ from ‘severe’ errors
- Alert triggers are created by hand
- Developers are human and make mistakes

Proposed Solution: Logness

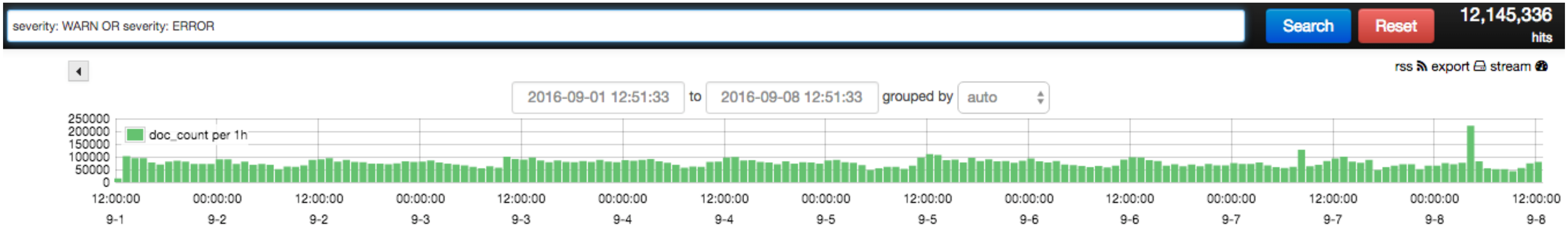
- Use fingerprint-based algorithm to combine error logs into abstract errors
- Ask devs to rate abstract errors (severe, monitored, non-severe)
- Filter, and focus on differences before and after (weekly) patches

- Implement on top of ELK stack

Error Grouping Algorithm



Original Kibana display:

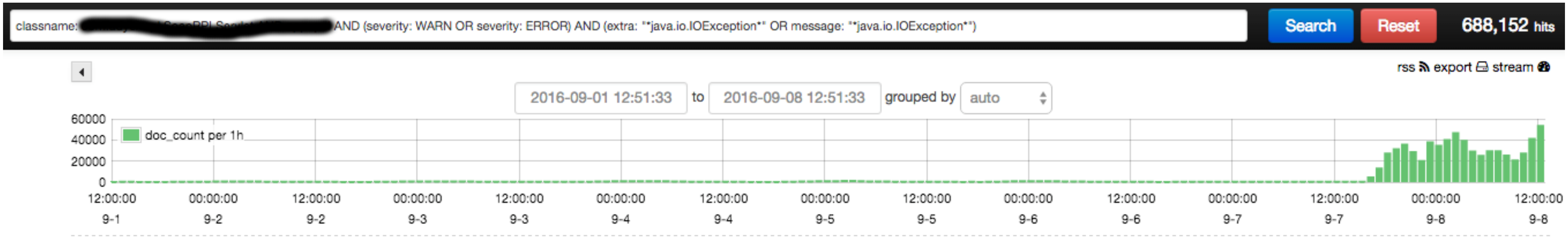


```
severity: WARN OR severity: ERROR
```

12,145,336 hits

2016-09-01 12:51:33 to 2016-09-08 12:51:33 grouped by auto

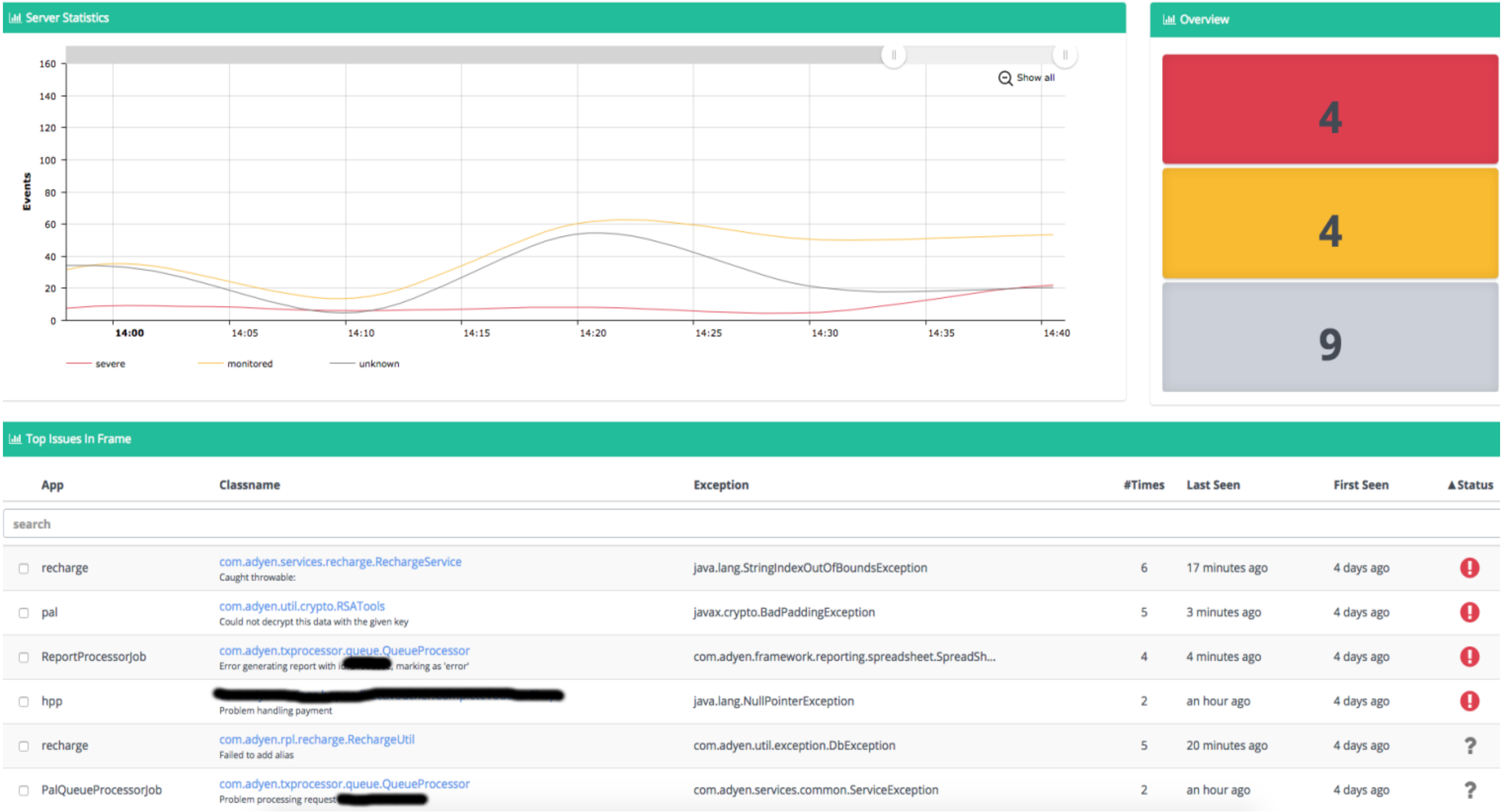
Filtering by Logness, with actual problem found



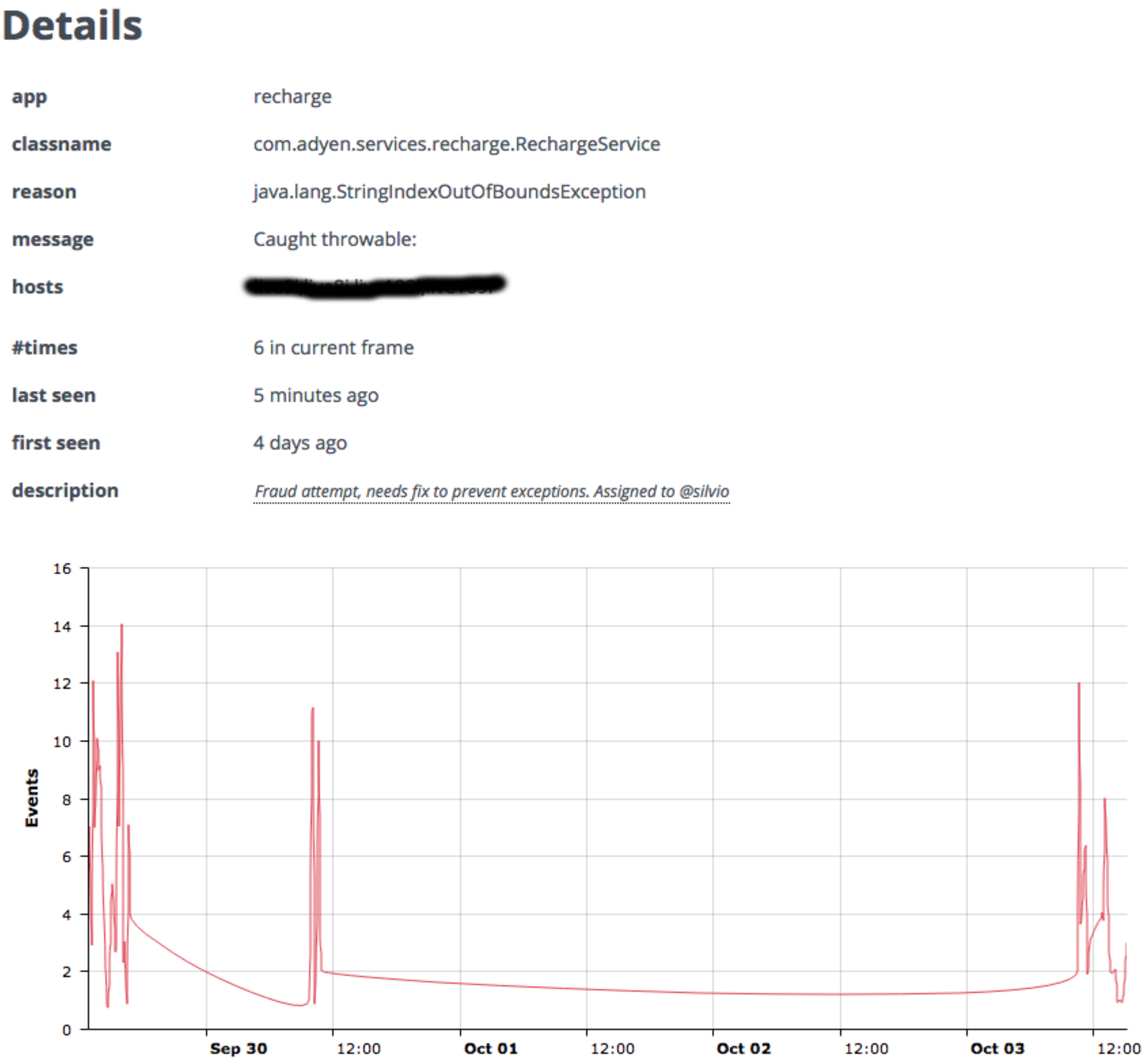
```
classname: [redacted] AND (severity: WARN OR severity: ERROR) AND (extra: "*java.io.IOException*" OR message: "*java.io.IOException*")
```

688,152 hits

2016-09-01 12:51:33 to 2016-09-08 12:51:33 grouped by auto



	App	Classname	Exception	#Times	Last Seen	First Seen	▲Status
	search						
□	recharge	com.adyen.services.recharge.RechargeService Caught throwable:	java.lang.StringIndexOutOfBoundsException	6	17 minutes ago	4 days ago	!
□	pal	com.adyen.util.crypto.RSATools Could not decrypt this data with the given key	javax.crypto.BadPaddingException	5	3 minutes ago	4 days ago	!
□	ReportProcessorJob	com.adyen.txprocessor.queue.QueueProcessor Error generating report with i[redacted], marking as 'error'	com.adyen.framework.reporting.spreadsheet.SpreadSh...	4	4 minutes ago	4 days ago	!
□	hpp	[redacted] Problem handling payment	java.lang.NullPointerException	2	an hour ago	4 days ago	!
□	recharge	com.adyen.rpl.recharge.RechargeUtil Failed to add alias	com.adyen.util.exception.DbException	5	20 minutes ago	4 days ago	?
□	PalQueueProcessorJob	com.adyen.txprocessor.queue.QueueProcessor Problem processing request[redacted]	com.adyen.services.common.ServiceException	2	an hour ago	4 days ago	?



Details

app	recharge
classname	com.adyen.services.recharge.RechargeService
reason	java.lang.StringIndexOutOfBoundsException
message	Caught throwable:
hosts	[redacted]
#times	6 in current frame
last seen	5 minutes ago
first seen	4 days ago
description	*Fraud attempt, needs fix to prevent exceptions. Assigned to @silvio*

Implications Logness

- Solved 9 different issues during development, including critical ones (hotfixes during a patch)
- Mainly by visualizing logs in a more structured way
- Developers are more aware and can quickly investigate errors on ‘their’ application

Bug Example

- China Released New Creditcard Numbers
- Starting with 95 ...
- Being 19 digits long
- Are stored as a LONG because of the ISO
- 9511111111111111111111 is a LONG overflow
- Impact: ... 4 people / month ... at the moment

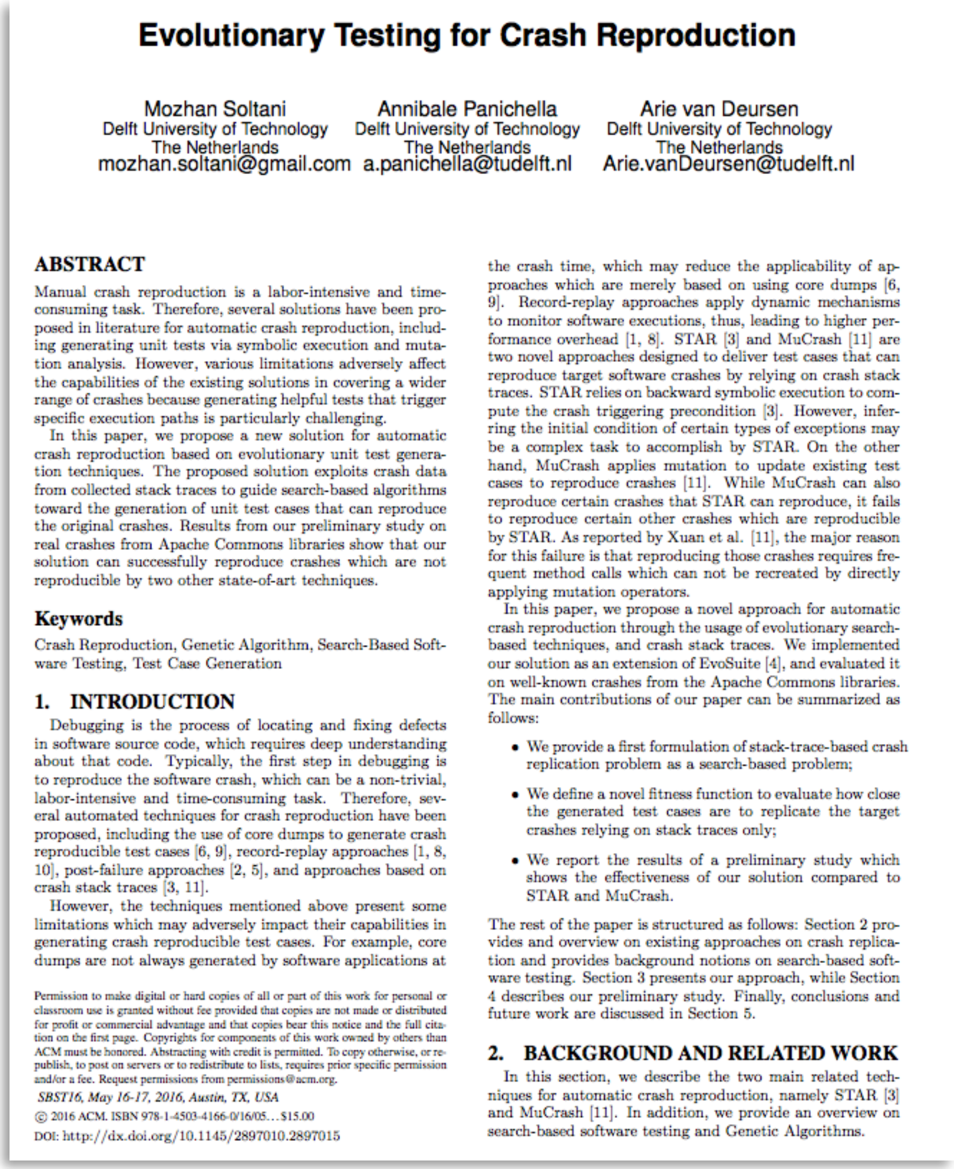
Logness: Summary

- Hundreds of thousands of errors and warnings in logs
- Group into common errors
- Focus on differences before/after patch
- Classify severity (machine learning)
- Visualize using Kibana

- Proven useful to find range of actual problems.

Part 4: Crash Reproduction

- Assume we found a stack trace of interest.
- How can we reproduce it so that we can fix it?
- How can we create a test case to avoid it from now on?



Evolutionary Testing for Crash Reproduction

Mozhan Soltani
Delft University of Technology
The Netherlands
mozhan.soltani@gmail.com

Annibale Panichella
Delft University of Technology
The Netherlands
a.panichella@tudelft.nl

Arie van Deursen
Delft University of Technology
The Netherlands
Arie.vanDeursen@tudelft.nl

ABSTRACT

Manual crash reproduction is a labor-intensive and time-consuming task. Therefore, several solutions have been proposed in literature for automatic crash reproduction, including generating unit tests via symbolic execution and mutation analysis. However, various limitations adversely affect the capabilities of the existing solutions in covering a wider range of crashes because generating helpful tests that trigger specific execution paths is particularly challenging.

In this paper, we propose a new solution for automatic crash reproduction based on evolutionary unit test generation techniques. The proposed solution exploits crash data from collected stack traces to guide search-based algorithms toward the generation of unit test cases that can reproduce the original crashes. Results from our preliminary study on real crashes from Apache Commons libraries show that our solution can successfully reproduce crashes which are not reproducible by two other state-of-art techniques.

Keywords

Crash Reproduction, Genetic Algorithm, Search-Based Software Testing, Test Case Generation

1. INTRODUCTION

Debugging is the process of locating and fixing defects in software source code, which requires deep understanding about that code. Typically, the first step in debugging is to reproduce the software crash, which can be a non-trivial, labor-intensive and time-consuming task. Therefore, several automated techniques for crash reproduction have been proposed, including the use of core dumps to generate crash reproducible test cases [6, 9], record-replay approaches [1, 8, 10], post-failure approaches [2, 5], and approaches based on crash stack traces [3, 11].

However, the techniques mentioned above present some limitations which may adversely impact their capabilities in generating crash reproducible test cases. For example, core dumps are not always generated by software applications at the crash time, which may reduce the applicability of approaches which are merely based on using core dumps [6, 9]. Record-replay approaches apply dynamic mechanisms to monitor software executions, thus, leading to higher performance overhead [1, 8]. STAR [3] and MuCrash [11] are two novel approaches designed to deliver test cases that can reproduce target software crashes by relying on crash stack traces. STAR relies on backward symbolic execution to compute the crash triggering precondition [3]. However, inferring the initial condition of certain types of exceptions may be a complex task to accomplish by STAR. On the other hand, MuCrash applies mutation to update existing test cases to reproduce crashes [11]. While MuCrash can also reproduce certain crashes that STAR can reproduce, it fails to reproduce certain other crashes which are reproducible by STAR. As reported by Xuan et al. [11], the major reason for this failure is that reproducing those crashes requires frequent method calls which can not be recreated by directly applying mutation operators.

In this paper, we propose a novel approach for automatic crash reproduction through the usage of evolutionary search-based techniques, and crash stack traces. We implemented our solution as an extension of EvoSuite [4], and evaluated it on well-known crashes from the Apache Commons libraries. The main contributions of our paper can be summarized as follows:

- We provide a first formulation of stack-trace-based crash replication problem as a search-based problem;
- We define a novel fitness function to evaluate how close the generated test cases are to replicate the target crashes relying on stack traces only;
- We report the results of a preliminary study which shows the effectiveness of our solution compared to STAR and MuCrash.

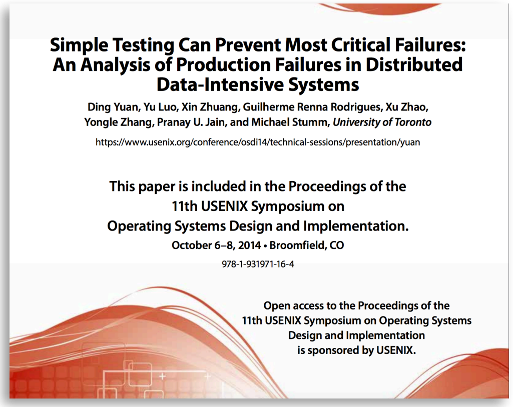
The rest of the paper is structured as follows: Section 2 provides and overview on existing approaches on crash replication and provides background notions on search-based software testing. Section 3 presents our approach, while Section 4 describes our preliminary study. Finally, conclusions and future work are discussed in Section 5.

2. BACKGROUND AND RELATED WORK

In this section, we describe the two main related techniques for automatic crash reproduction, namely STAR [3] and MuCrash [11]. In addition, we provide an overview on search-based software testing and Genetic Algorithms.

SBST16, May 16-17, 2016, Austin, TX, USA
© 2016 ACM. ISBN 978-1-4503-4166-0/16/05…$15.00
DOI: http://dx.doi.org/10.1145/2897010.2897015

Crash Reproduction: Feasibility



Simple Testing Can Prevent Most Critical Failures: An Analysis of Production Failures in Distributed Data-Intensive Systems

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao,
Yongle Zhang, Pranay U. Jain, and Michael Stumm, *University of Toronto*

https://www.usenix.org/conference/osdi14/technical-sessions/presentation/yuan

This paper is included in the Proceedings of the
11th USENIX Symposium on
Operating Systems Design and Implementation.
October 6–8, 2014 • Broomfield, CO
978-1-931971-16-4

Open access to the Proceedings of the
11th USENIX Symposium on Operating Systems
Design and Implementation
is sponsored by USENIX.

- 74% of the failures are deterministic: guaranteed to manifest given the right input event sequences.
- 76% of the failures print explicit failure related error messages.
- For majority (84%) of failures, all triggering events are logged.
- A majority of the production failures (77%) can be reproduced by a unit test.

- *Let’s automate this.*

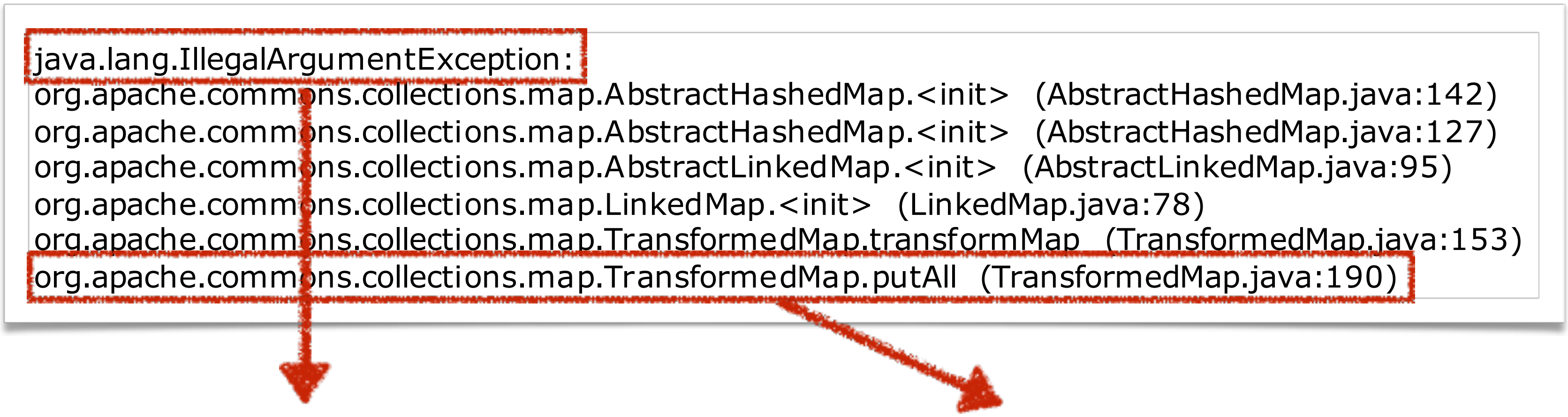
Anatomy of a Stack Trace

Target Crash

Bug Name: ACC-48

Library: Apache Commons Collection

Stack Trace



```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init>  (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init>  (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init>  (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init>  (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap  (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll  (TransformedMap.java:190)
```

Exception Name

Root Cause of the Exception

https://issues.apache.org/jira/browse/COLLECTIONS-48

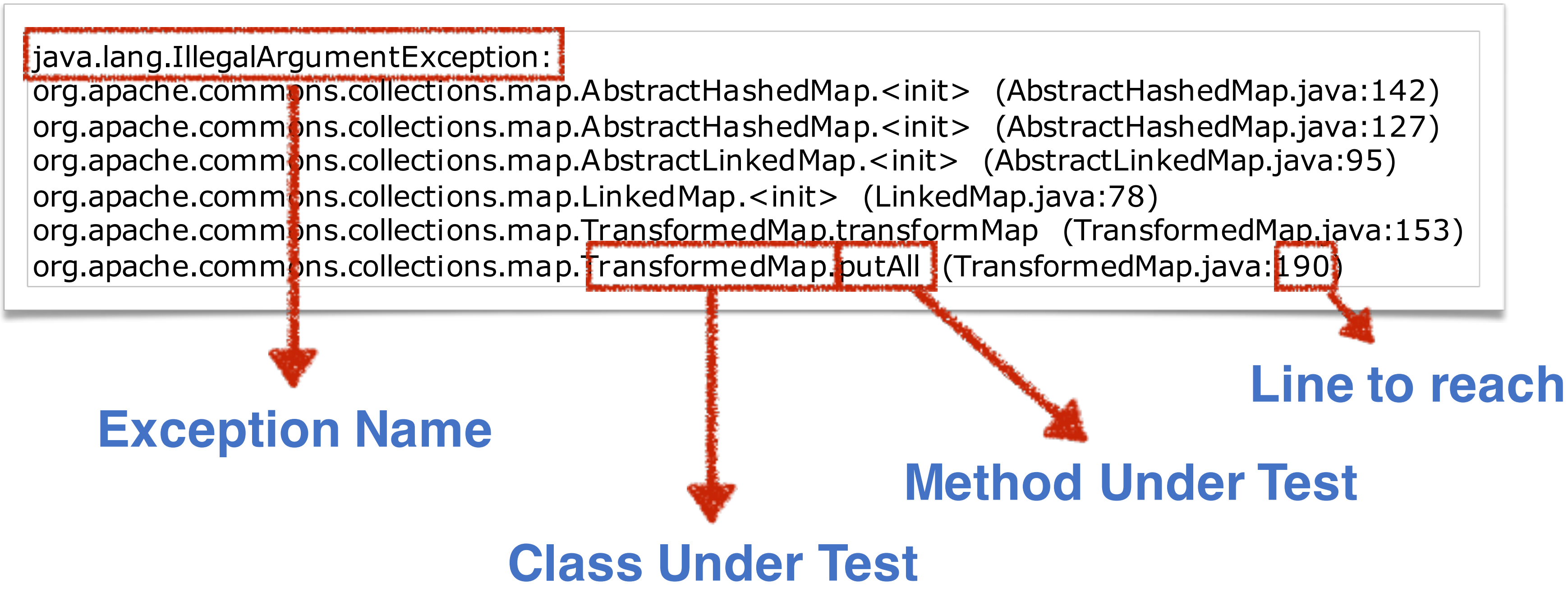
Anatomy of a Stack Trace

Target Crash

Bug Name: ACC-48

Library: Apache Commons Collection

Stack Trace



```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init>  (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init>  (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init>  (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init>  (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap  (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll  (TransformedMap.java:190)
```

Exception Name

Class Under Test

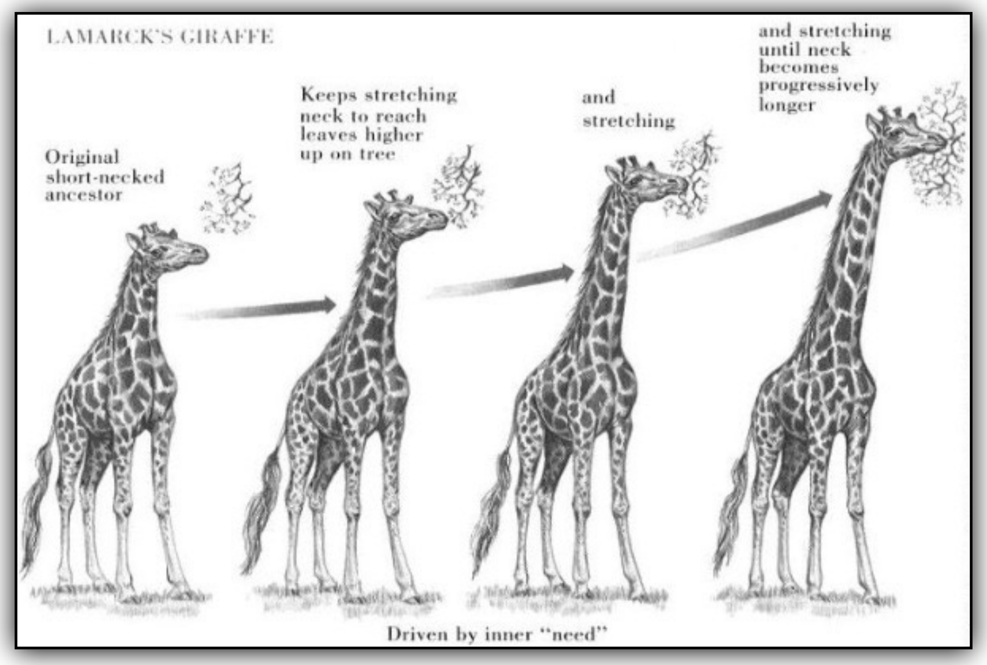
Method Under Test

Line to reach

https://issues.apache.org/jira/browse/COLLECTIONS-48

Genetic Algorithms

Genetic Algorithm: search algorithm inspired be evolution theory


LAMARCK'S GIRAFFE
Original short-necked ancestor
Keeps stretching neck to reach leaves higher up on tree
and stretching
and stretching until neck becomes progressively longer
Driven by inner "need"

1. Natural selection: Individuals that best fit the natural environment survive (worst die)
2. Reproduction: survived individuals generate offsprings (next generation)
3. Mutation: offsprings inherits properties of their parents (with some mutations)
4. Iteration: generation by generation the new offspring fit better the environment than their parents

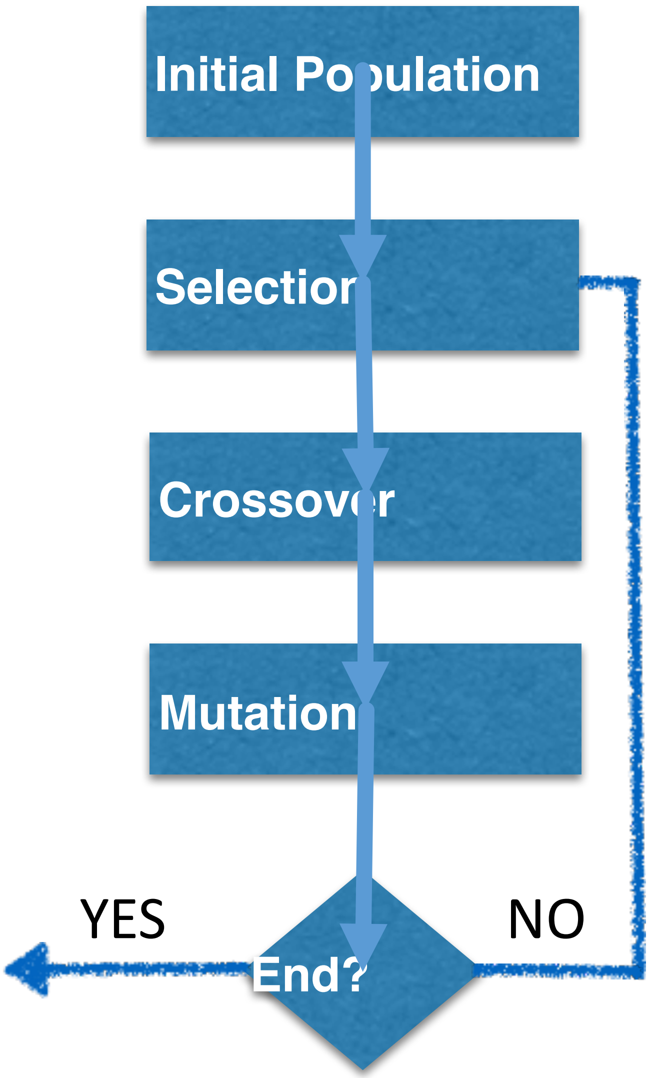
Automated Test Case Generation

Generation of test cases

1. Select one statement (target)
2. Using genetic algorithm to search for method calls and input parameters that allow to cover the selected target
3. Store the test case
4. Repeat steps 1-4 until all statements are covered

Goal-oriented
or
Single-target

Genetic Algorithms



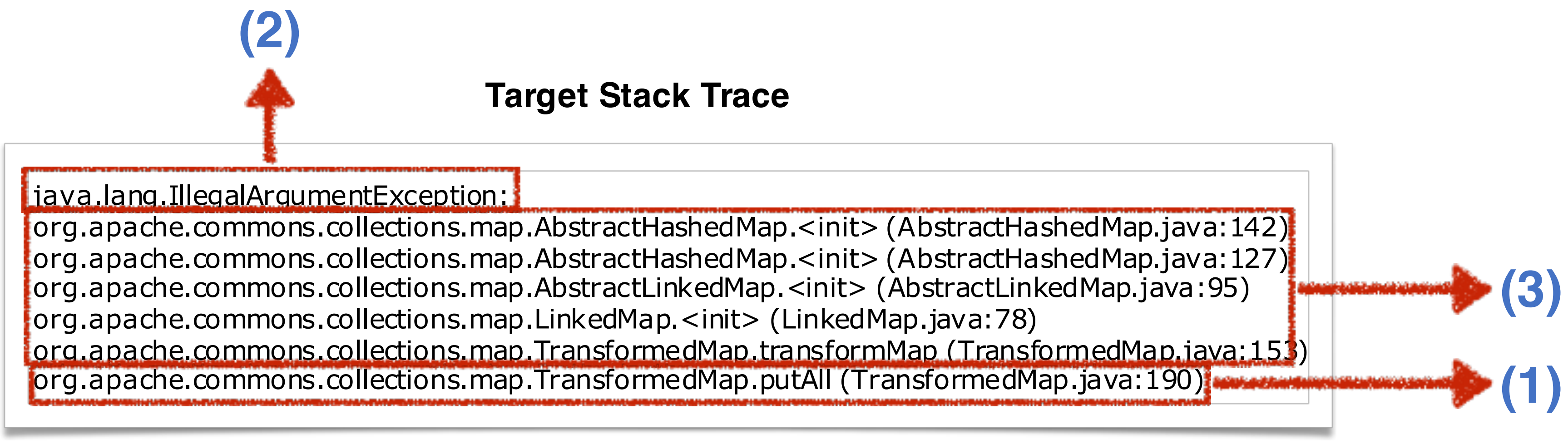
EVOSUITE

- Tool to automatically generate JUnit test suite using genetic algorithm
- Generate and optimize test suite to work towards satisfying a given coverage criterion
- Generate assertions concisely documenting current behavior (oracle)
  - To test future versions that should keep this behavior

Fitness Function

<u>Main Conditions to Satisfy</u>

1) the line (statement) where the exception is thrown has to be covered
2) the target exception has to be thrown
3) the generated stack trace must be as similar to the original one as possible.



Target Stack Trace

```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init> (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init> (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll (TransformedMap.java:190)
```

$$f(t) = 3 \text{ x } \underbrace{\text{line\_coverage}}_{(1)} + 2 \text{ x } \underbrace{\text{exception\_coverage}}_{(2)} + \underbrace{\text{trace\_similarity}}_{(3)}$$

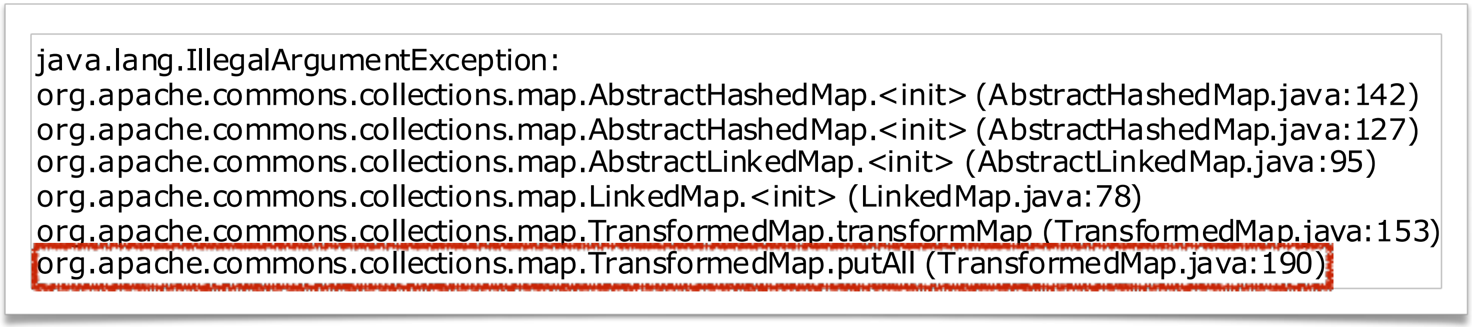
Fitness Function

f(t) **=** 3 x line_coverage + 2 x exception_coverage + trace_similarity

(1) **(2)** **(3)**

1) **line_coverage** = approach_level + branch_distance

Target Stack Trace



```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init> (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init> (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll (TransformedMap.java:190)
```

Fitness Function

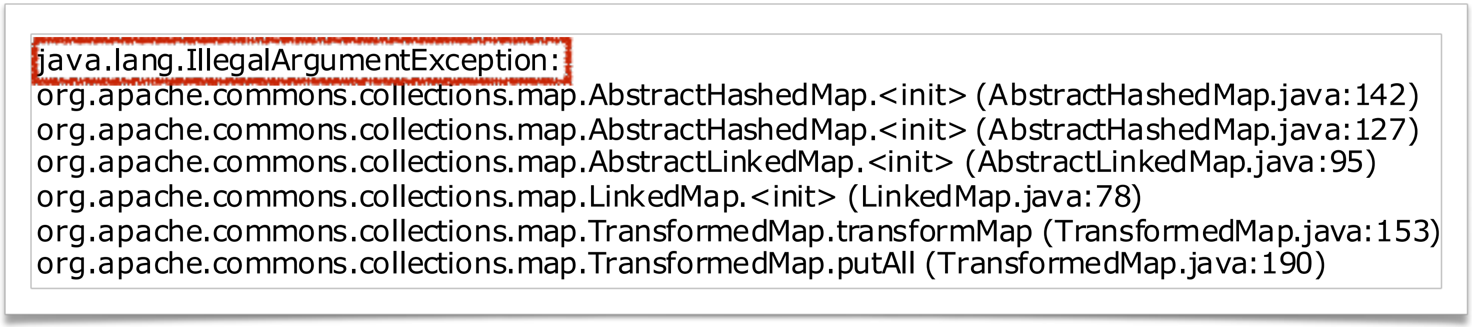
$f(t)$ **=** 3 x line_coverage + 2 x exception_coverage + trace_similarity

(1) **(2)** **(3)**

1) **line_coverage** = approach_level + branch_distance

2) **exception_coverage** = 0 if the target exception in thrown; 1 otherwise

Target Stack Trace



```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init> (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init> (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll (TransformedMap.java:190)
```

Fitness Function

***f(t)* = 3 x line_coverage + 2 x exception_coverage + trace_similarity**

(1) (2) (3)

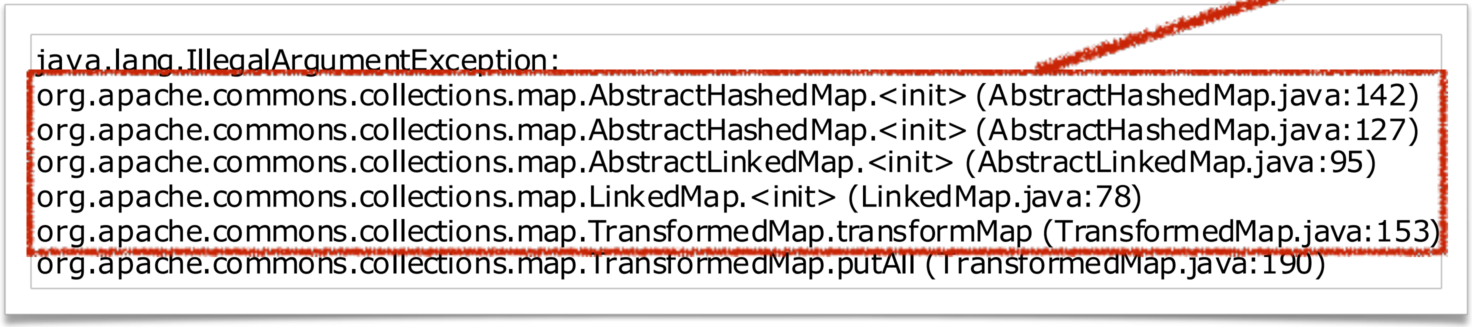
1) **line_coverage** = approach_level + branch_distance

2) **exception_coverage** = 0 if the target exception in thrown; 1 otherwise

3) **trace_similarity** = class name, method name, triggering line

Trace Elements

Target Stack Trace



```
java.lang.IllegalArgumentException:
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:142)
org.apache.commons.collections.map.AbstractHashedMap.<init> (AbstractHashedMap.java:127)
org.apache.commons.collections.map.AbstractLinkedMap.<init> (AbstractLinkedMap.java:95)
org.apache.commons.collections.map.LinkedMap.<init> (LinkedMap.java:78)
org.apache.commons.collections.map.TransformedMap.transformMap (TransformedMap.java:153)
org.apache.commons.collections.map.TransformedMap.putAll (TransformedMap.java:190)
```

Empirical Evaluation

Context: 10 real bugs from Apache Commons Collections

ACC-4	2.0	NullPointer	Major
ACC-28	2.0	NullPointer	Major
ACC-35	2.0	UnsupportedOperation	Major
ACC-48	3.1	IllegalArgument	Major
ACC-53	3.1	ArrayIndexOutOfBound	Major
ACC-70	3.1	NullPointer	Major
ACC-77	3.1	IllegalState	Major
ACC-104	3.1	ArrayIndexOutOfBound	Major
ACC-331	3.2	NullPointer	Minor
ACC-377	3.2	NullPointer	Minor

Used in:

N. Chen and Kim, TSE 2015.

J. Xuan et al., ESEC/FSE 2015.

Experimented algorithms:

- EvoSuite + our fitness function (30 independent runs)
- STAR (Symbolic execution)
- MuCrash (Mutation analysis)

Results

Bug	**% Successful Replication**	**STAR**	**MuCrash**
ACC-4	30/30	YES	YES
ACC-28	30/30	YES	YES
ACC-35	30/30	YES	YES
ACC-48	30/30	YES	YES
ACC-53	28/30	YES	NO
ACC-70	30/30	NO	NO
ACC-77	30/30	YES	NO
ACC-104	0/30	YES	YES
ACC-331	10/30	NO	YES
ACC-377	0/30	NO	NO

Results

Bug ID	% Successful Replication	STAR	MuCrash
ACC-4	30/30	YES	YES
ACC-28	30/30	YES	YES
ACC-35	30/30	YES	YES
ACC-48	30/30	YES	YES
ACC-53	28/30	YES	NO
ACC-70	30/30	NO	NO
ACC-77	30/30	YES	NO
ACC-104	0/30	YES	YES
ACC-331	10/30	NO	YES
ACC-377	0/30	NO	NO

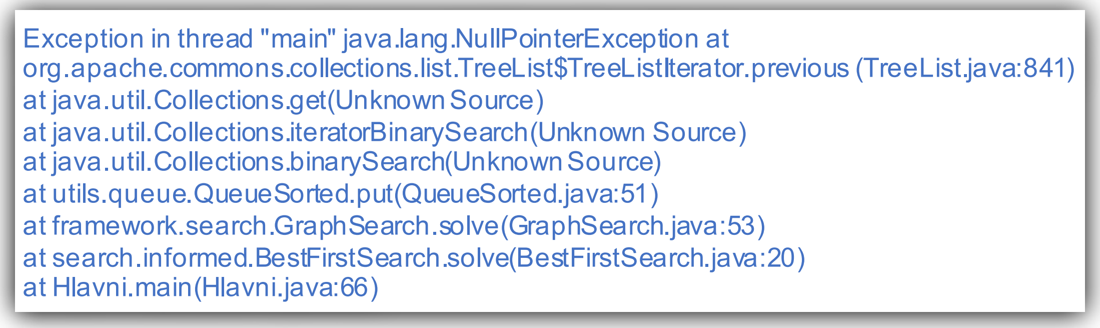
Our solution replicated 8/10 bugs

STAR replicated 7/10 bugs

MuCrash replicated 6/10 bugs

ACC-70

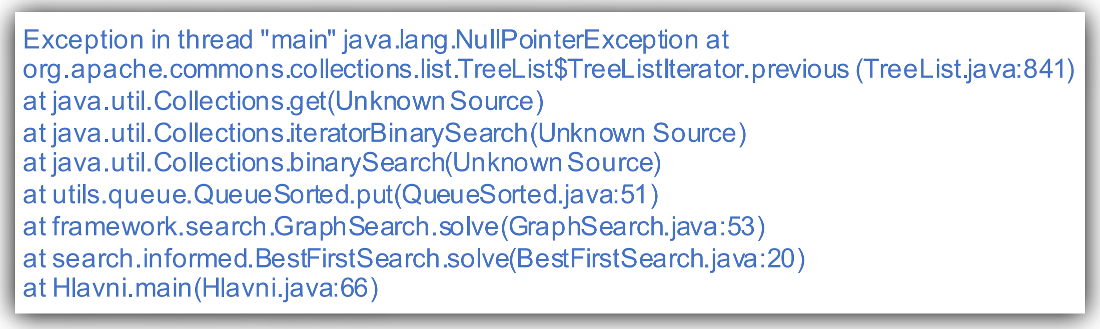
Target Stack Trace



```
Exception in thread "main" java.lang.NullPointerException at
org.apache.commons.collections.list.TreeList$TreeListIterator.previous (TreeList.java:841)
at java.util.Collections.get(Unknown Source)
at java.util.Collections.iteratorBinarySearch(Unknown Source)
at java.util.Collections.binarySearch(Unknown Source)
at utils.queue.QueueSorted.put(QueueSorted.java:51)
at framework.search.GraphSearch.solve(GraphSearch.java:53)
at search.informed.BestFirstSearch.solve(BestFirstSearch.java:20)
at Hlavni.main(Hlavni.java:66)
```

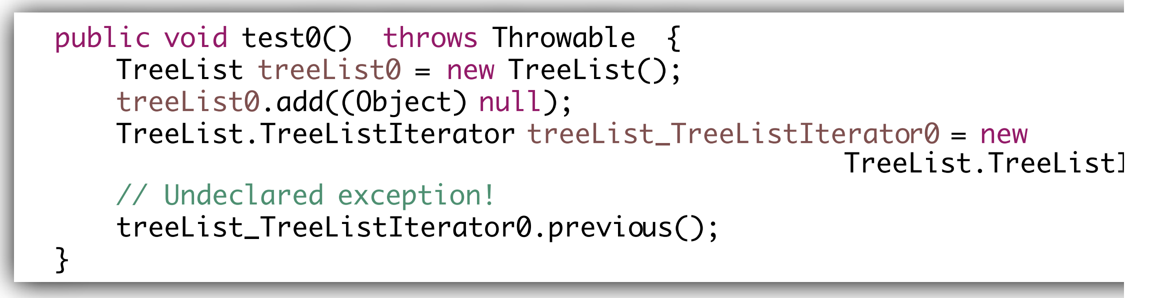
ACC-70

Target Stack Trace



```
Exception in thread "main" java.lang.NullPointerException at
org.apache.commons.collections.list.TreeList$TreeListIterator.previous (TreeList.java:841)
at java.util.Collections.get(Unknown Source)
at java.util.Collections.iteratorBinarySearch(Unknown Source)
at java.util.Collections.binarySearch(Unknown Source)
at utils.queue.QueueSorted.put(QueueSorted.java:51)
at framework.search.GraphSearch.solve(GraphSearch.java:53)
at search.informed.BestFirstSearch.solve(BestFirstSearch.java:20)
at Hlavni.main(Hlavni.java:66)
```

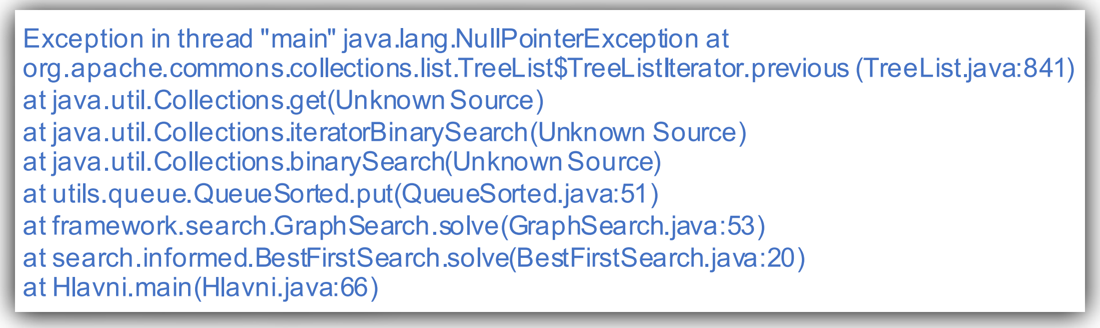
Test generated by our solution



```
public void test0()  throws Throwable  {
    TreeList treeList0 = new TreeList();
    treeList0.add((Object) null);
    TreeList.TreeListIterator treeList_TreeListIterator0 = new
                                               TreeList.TreeListI
    // Undeclared exception!
    treeList_TreeListIterator0.previous();
}
```

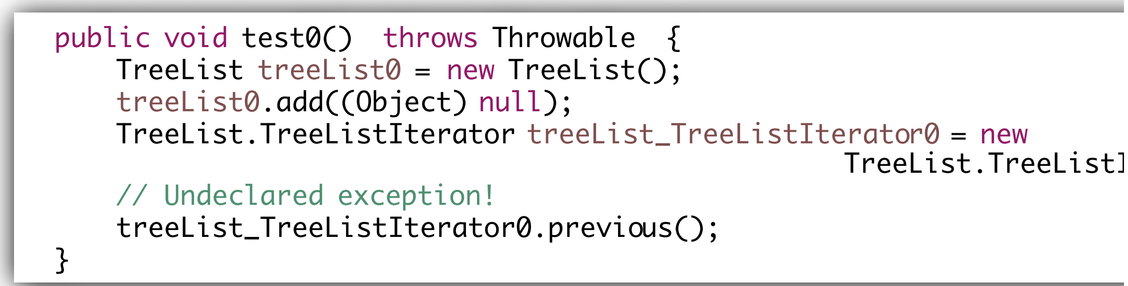
ACC-70

Target Stack Trace



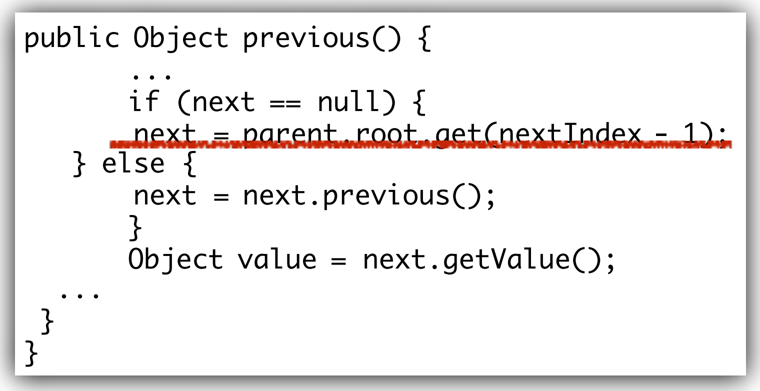
```
Exception in thread "main" java.lang.NullPointerException at
org.apache.commons.collections.list.TreeList$TreeListIterator.previous (TreeList.java:841)
at java.util.Collections.get(Unknown Source)
at java.util.Collections.iteratorBinarySearch(Unknown Source)
at java.util.Collections.binarySearch(Unknown Source)
at utils.queue.QueueSorted.put(QueueSorted.java:51)
at framework.search.GraphSearch.solve(GraphSearch.java:53)
at search.informed.BestFirstSearch.solve(BestFirstSearch.java:20)
at Hlavni.main(Hlavni.java:66)
```

Test generated by our solution



```
public void test0()  throws Throwable  {
    TreeList treeList0 = new TreeList();
    treeList0.add((Object) null);
    TreeList.TreeListIterator treeList_TreeListIterator0 = new
                                                    TreeList.TreeListI
    // Undeclared exception!
    treeList_TreeListIterator0.previous();
}
```

Affected Code



```
public Object previous() {
       ...
       if (next == null) {
        next = parent.root.get(nextIndex - 1);
   } else {
        next = next.previous();
       }
       Object value = next.getValue();
    ...
  }
}
```

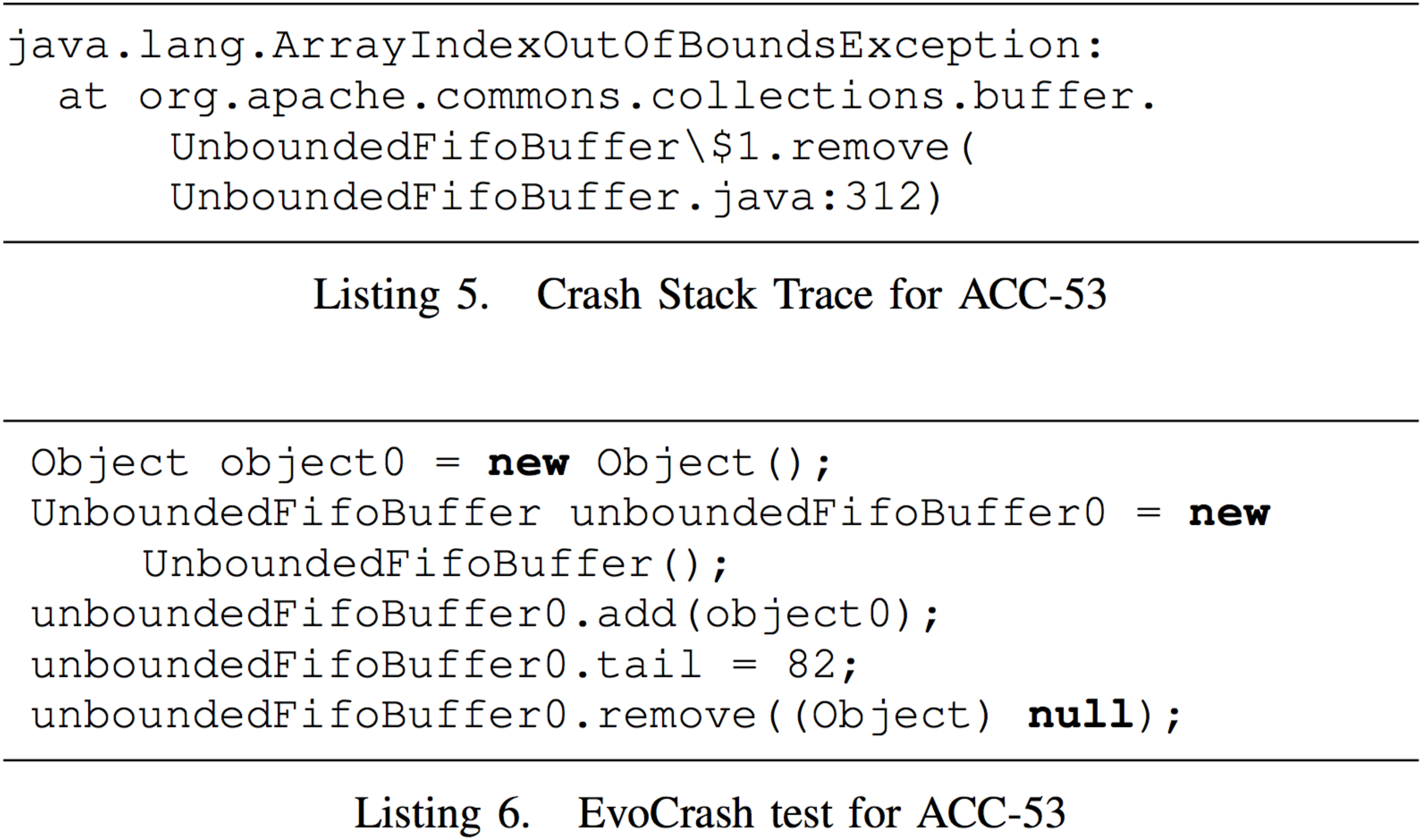
if "parent" is null, this code generates an exception

```
java.lang.NullPointerException:
  at org.apache.tools.ant.util.SymbolicLinkUtils.
      isSymbolicLink(SymbolicLinkUtils.java:107)
  at org.apache.tools.ant.util.SymbolicLinkUtils.
      isSymbolicLink(SymbolicLinkUtils.java:73)
  at org.apache.tools.ant.util.SymbolicLinkUtils.
      deleteSymbolicLink(SymbolicLinkUtils.java:223)
  at org.apache.tools.ant.taskdefs.optional.unix.
      Symlink.delete(Symlink.java:187)
```

Listing 1. Crash Stack Trace for ANT-49137.

```
 public void test0()  throws Throwable  {
  Symlink symlink0 = new Symlink();
  symlink0.setLink("");
  symlink0.delete();
}
```

Listing 2. Generated test by EvoCrash for ANT-49137.



```
java.lang.ArrayIndexOutOfBoundsException:
  at org.apache.commons.collections.buffer.
      UnboundedFifoBuffer\$1.remove(
      UnboundedFifoBuffer.java:312)
```

Listing 5. Crash Stack Trace for ACC-53

```
Object object0 = new Object();
UnboundedFifoBuffer unboundedFifoBuffer0 = new
    UnboundedFifoBuffer();
unboundedFifoBuffer0.add(object0);
unboundedFifoBuffer0.tail = 82;
unboundedFifoBuffer0.remove((Object) null);
```

Listing 6. EvoCrash test for ACC-53

[Under review]

- New “guided” genetic algorithm
- Better fitness function
- Case studies: Apache ant, commons, log4j
- 38 out of 50 crashes replicated

A Guided Genetic Algorithm for Automated Crash Reproduction

Mozhan Soltani
Delft University of Technology
The Netherlands
m.soltani@tudelft.nl

Annibale Panichella
Delft University of Technology
The Netherlands
a.panichella@tudelft.nl

Arie van Deursen
Delft University of Technology
The Netherlands
Arie.vanDeursen@tudelft.nl

***Abstract*—To reduce the effort developers have to make for crash debugging, researchers have proposed several solutions for automatic failure reproduction. Recent advances proposed the usage of symbolic execution, mutation analysis and directed model checking as underling techniques for post-failure analysis of crash stack traces. However, existing approaches still cannot reproduce many real-world crashes due to various limitations, such as environment dependencies, path explosion, and time complexity. In this paper, we present EvoCrash, a post-failure approach which uses a novel Guided Genetic Algorithm (GGA) to cope with the large search space characterizing real-world software programs, and thereby address major challenges in automated crash replication. Results of an empirical study on three open-source systems show that EvoCrash can successfully replicate 33 (66%) of real-world crashes, thereby outperforming the three cutting-edge crash replication techniques.**

***Keywords*-Search-Based Software Testing; Genetic Algorithms; Automated Crash Reproduction;**

I. Introduction

Manual crash replication is a labor-intensive task. Developers faced with this task need to reproduce failures reported in issue tracking systems, which all too often contain insufficient data to determine the root cause of a failure.

Hence, to reduce developer effort, many different automated crash replication techniques have been proposed in the literature. Such techniques typically aim at generating tests triggering the target failure. For example, record-replay approaches [1]–[5] monitor software behavior via software/hardware instrumentation to collect the observed objects and method calls when failures occur. Unfortunately, such techniques suffer from well-known practical limitations, such as performance overhead [6], and privacy issues [7].

As opposed to these costly techniques, *post-failure* approaches [6]–[12] try to replicate crashes by exploiting data that is available *after* the failure, typically stored in log files or external bug tracking systems. Most of these techniques require specific input data in addition to crash stack traces [6], such as core dumps [8]–[10], [13] or models of the software like input grammars [14], [15] or class invariants [16].

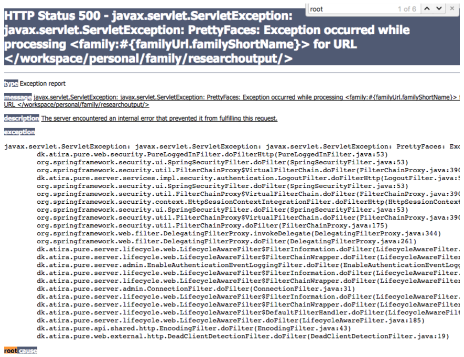
Since such additional information is usually not available to developers, recent advances in the field have focused on crash stack traces as the *only* source of information for debugging [6], [7], [12]. For example, Chen and Kim developed STAR [6], an approach based on backward symbolic execution. STAR outperforms earlier crash replication techniques, such as Randoop [17] and BugRedux [18]. Xuan et al. [12] presented MuCrash, a tool that updates existing test cases using specific mutation operators, thus creating a new pool of tests to run against the software under test. Nayrolle et al. [7] proposed JCHARMING, based on directed model checking combined with program slicing [7], [19].

Unfortunately, the state-of-the-art tools suffer from several limitations. For example, STAR cannot handle cases with external environment dependencies [6] (e.g., file or network inputs), non-trivial string constraints, or complex logic potentially leading to a path explosion. MuCrash is limited by the ability of existing tests in covering method call sequences of interest, and it may lead to a large number of unnecessary mutated test cases [12]. JCHARMING [7], [19] applies model checking which can be computationally expensive. Moreover, similar to STAR, JCHARMING does not handle crash cases with environmental dependencies.

In our previous preliminary study [20], we have suggested to re-use existing unit test generation tools, such as EvoSuite [21], for crash replication. To that end, we developed a *fitness function* to assess the capability of candidate test cases in replicating the target failure. Although this simple solution could help to replicate one crash not handled by STAR and MuCrash, our preliminary study showed that this simple solution still leaves other crashes as non-reproducible. These negative results are due to the large search space for real world programs where the probability to generate test data satisfying desired failure conditions is low. In fact, the classic *genetic operators* from existing test frameworks are aimed at maximizing specific coverage criteria [21] instead of exploiting single execution paths and object states that characterize software failures.

To address this challenge, this paper presents an evolutionary search-based approach, named EvoCrash, for crash reproduction. EvoCrash is built on top of EvoSuite [21], the well-known automatic test suite generation tool for Java. For EvoCrash we developed a novel *guided* genetic algorithm (GGA). It lets the stack trace guide the search, thus reducing the search space. In particular, GGA uses a novel generative routine to build an initial population of tests exercising at least one of the methods reported in the crash stack frames.

Big Software on the Run



HTTP Status 500 - javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exception occurred while processing <family:#{familyUrl.familyShortName}> for URL </workspace/personal/family/researchoutput/>

type Exception report

message javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exception occurred while processing <family:#{familyUrl.familyShortName}> URL </workspace/personal/family/researchoutput/>

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
javax.servlet.ServletException: javax.servlet.ServletException: javax.servlet.ServletException: PrettyFaces: Exc
	dk.atira.pure.web.security.PureLoggedInFilter.doFilterHttp(PureLoggedInFilter.java:53)
	org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
	org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39
	dk.atira.pure.server.services.impl.security.authentication.LogoutFilter.doFilterHttp(LogoutFilter.java:
	org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
	org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39
	org.springframework.security.context.HttpSessionContextIntegrationFilter.doFilterHttp(HttpSessionContext
	org.springframework.security.ui.SpringSecurityFilter.doFilter(SpringSecurityFilter.java:53)
	org.springframework.security.util.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:39
	org.springframework.security.util.FilterChainProxy.doFilter(FilterChainProxy.java:175)
	org.springframework.web.filter.DelegatingFilterProxy.invokeDelegate(DelegatingFilterProxy.java:344)
	org.springframework.web.filter.DelegatingFilterProxy.doFilter(DelegatingFilterProxy.java:261)
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilte
	dk.atira.pure.server.admin.EnableAuthenticationEventLoggingFilter.doFilter(EnableAuthenticationEventLog
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilte
	dk.atira.pure.server.admin.ConnectionFilter.doFilter(ConnectionFilter.java:31)
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterInformation.doFilter(LifecycleAwareFilter.
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$FilterChainWrapper.doFilter(LifecycleAwareFilte
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter$DefaultFilterHandler.doFilter(LifecycleAwareFil
	dk.atira.pure.server.lifecycle.web.LifecycleAwareFilter.doFilter(LifecycleAwareFilter.java:185)
	dk.atira.pure.api.shared.http.EncodingFilter.doFilter(EncodingFilter.java:43)
	dk.atira.pure.web.external.http.DeadClientDetectionFilter.doFilter(DeadClientDetectionFilter.java:19)
```

root cause

- Long running (distributed) systems
  - *All situations will occur*
  - *All exceptions will fire*

- How do developers deal with exceptions?
- How do exceptions manifest themselves in bugs?
- How do exceptions show up in log data?
- How can logged exceptions be reproduced?

Exceptional Logging: Key Points

- Error handling itself is error prone
- Developers work around Java’s checked exceptions
- Support is needed to find relevant exceptions in abundance of log data
- Automated crash reproduction is within reach

Rethinking Exception Handling?

- Empirical:
  How big is the exception handling problem?
- Software engineering:
  How can we deal with the EH problem?
- Language engineering:
  Can we think of new languages solving EH?
- Domain analysis: Anomaly versus exception?
  Can we push exceptions to the domain level?

TU Delft is Hiring!

- PhD position in EU STAMP project on test amplification
- Postdoc position in BSR project on exceptional logging.
- Tenure track positions in distributed systems, cyber security, algorithms

Image credit: goodsonsallterrainlogginginc.com