

# **Unveiling Exception Handling Bug Hazards in Android based on GitHub and Google Code Issues**

Roberta Coelho, Lucas Almeida,  
Georgios Gousios, Arie van Deursen

# GitHubとGoogle Code Issuesに基づいて Androidのバグを処理している例外を 調べてみた

Roberta Coelho, Lucas Almeida,  
Georgios Gousios, Arie van Deursen

# 例外

- Javaなどの言語では**実行時エラー**のことを**例外**と呼ぶ
- 例外発生時は**緊急用の処理**（**例外処理**）を行える
- 例外が発生すると**例外に関する情報を持ったオブジェクト**（**例外オブジェクト**）が生成される

# Javaの例外

## 検査例外

例外処理を書かないとコンパイルエラーになる

## 非検査例外

例外処理を書かなくてもコンパイルエラーにならない

# 例外オブジェクト

## Throwableクラス

基本の例外クラス。他の例外クラスはこれを継承する

## Errorクラス

システムが発生源のエラー（致命的）を扱う例外クラス

## Exceptionクラス（検査例外）

Java側で処理できるエラーを扱う例外クラス

## RuntimeExceptionクラス（非検査例外）

例外処理が必須ではない例外クラス

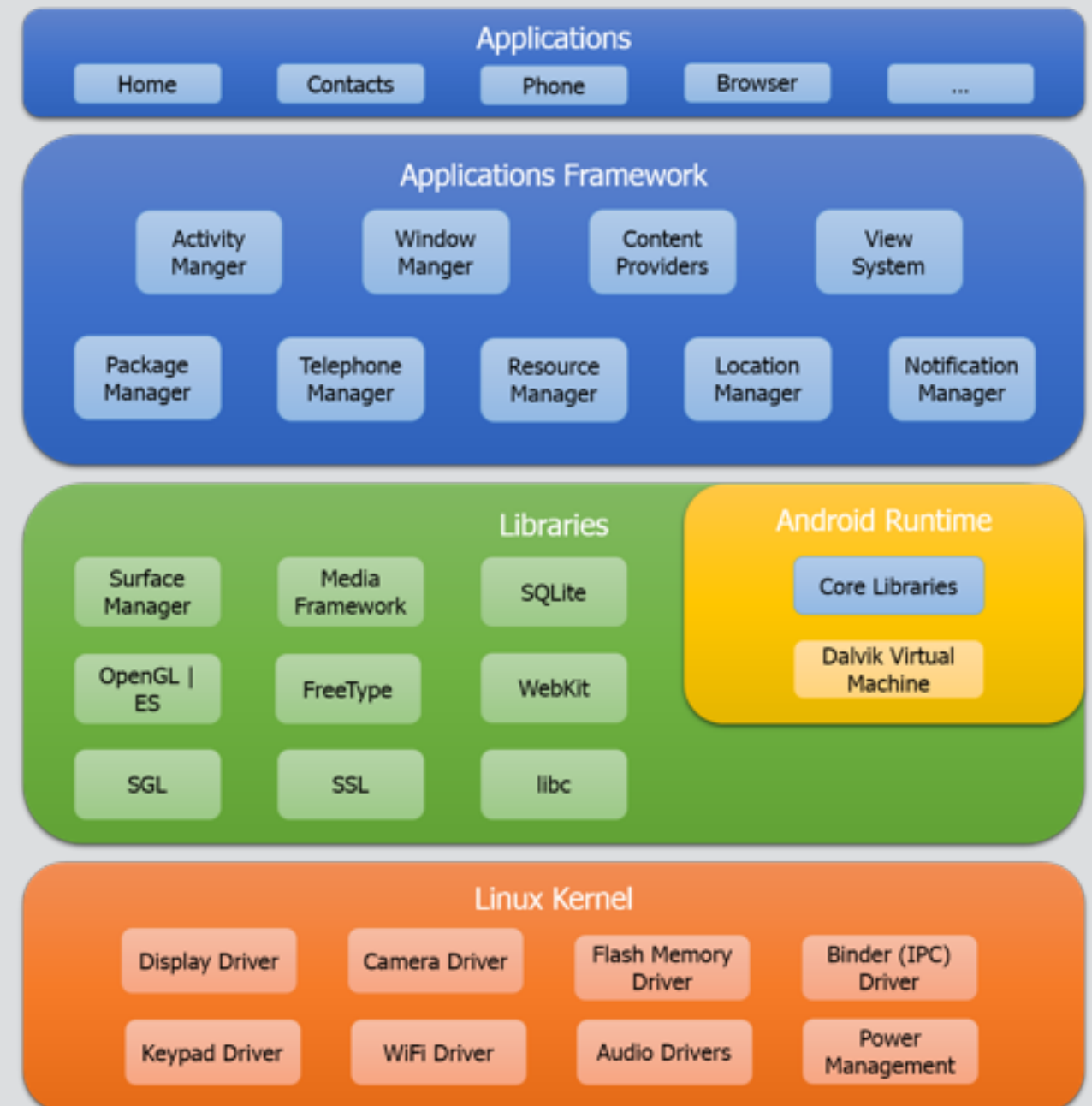
# 例外の伝播

例外が発生したとき、例外オブジェクトを**メソッドの呼び出し元**に渡してエラーを通知すること

# Android

各層が上層にAPIを提供する

C/C++で書かれたライブラリもある



# スタックトレース

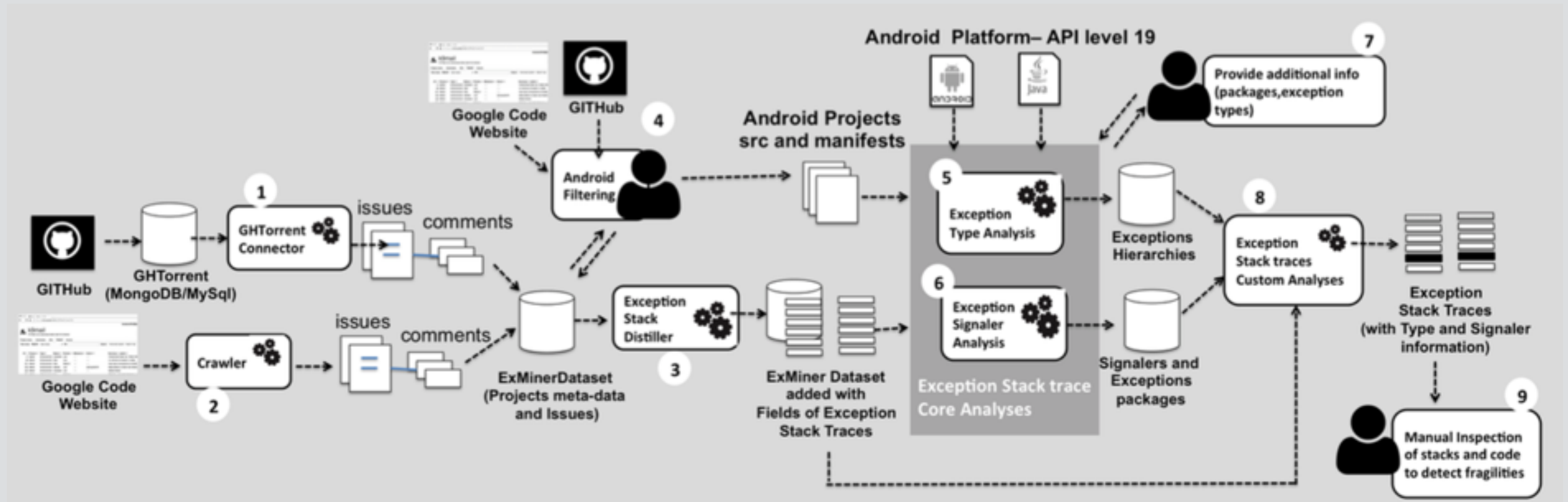
例外発生時に実行していたメソッドなどを表示したもの

# 研究の目的

スタックトレースがアプリケーションと下位のフレームワークの両方のバグを明らかにできるか調べること

今回のスタックトレースはAndroidプロジェクトのissuesに含まれるものを用いるが、他のJavaプロジェクトにもより深い考察を提供する

# 研究の流れ



- GitHubとGoogle Code Issuesからバグレポートを集める
- バグレポートからスタックトレースを抽出する
- スタックトレースと、ソースコードとバイトコードを関連付ける

# 研究結果（1）

| Root Exception                            | Projects |        | Occurrences |        | Android | Libcore | App | Lib |
|-------------------------------------------|----------|--------|-------------|--------|---------|---------|-----|-----|
|                                           | #        | %      | #           | %      |         |         |     |     |
| java.lang.NullPointerException            | 332      | 51.96% | 1664        | 27.71% | 525     | 20      | 836 | 280 |
| java.lang.IllegalStateException           | 120      | 18.78% | 278         | 4.63%  | 185     | 31      | 41  | 39  |
| java.lang.IllegalArgumentException        | 142      | 22.22% | 353         | 5.88%  | 195     | 12      | 95  | 44  |
| java.lang.RuntimeException                | 122      | 19.09% | 319         | 5.31%  | 203     | 2       | 64  | 51  |
| java.lang.OutOfMemoryError                | 78       | 12.21% | 237         | 3.95%  | 141     | 16      | 35  | 34  |
| java.lang.NoClassDefFoundError            | 67       | 10.49% | 94          | 1.57%  | 10      | 0       | 46  | 37  |
| java.lang.ClassCastException              | 64       | 10.02% | 130         | 2.16%  | 55      | 0       | 55  | 20  |
| java.lang.IndexOutOfBoundsException       | 62       | 9.70%  | 166         | 2.76%  | 53      | 0       | 93  | 18  |
| java.lang.NoSuchMethodError               | 54       | 8.45%  | 80          | 1.33%  | 10      | 0       | 56  | 14  |
| java.util.ConcurrentModificationException | 43       | 6.73%  | 65          | 1.08%  | 5       | 0       | 46  | 13  |

TABLE III: Root Exceptions occurrences and popularity in repositories hosted in Google Code (GC) and GitHub(GH).

例外処理を書かない状態で発生した例外の中では、  
NullPointerExceptionの発生率が最も高かった  
(プログラマが防げるはずの例外)

# 研究結果 (2)

| Concern                                | % Occurrences on stacks |
|----------------------------------------|-------------------------|
| Programming logic (java.lang and util) | 52,0%                   |
| Resources (IO, Memory, Batory)         | 23,9%                   |
| Security                               | 4,1%                    |
| Concurrency                            | 2,9%                    |
| Backward compatibility                 | 5,5%                    |
| Specific Exceptions                    | 4,9%                    |
| General (Error, Exception, Runtime)    | 6,7%                    |

TABLE IV: Identifying the concerns related to root exceptions

例外処理を書かない状態で発生した例外のうち  
50%以上がプログラミングのエラーに起因し、  
残りのうち25%が資源の制約に起因していた

# 例外オブジェクト

## Throwableクラス

基本の例外クラス。他の例外クラスはこれを継承する

## Errorクラス

システムが発生源のエラー（致命的）を扱う例外クラス

## Exceptionクラス（検査例外）

Java側で処理できるエラーを扱う例外クラス

## RuntimeExceptionクラス（非検査例外）

例外処理が必須ではない例外クラス

# 研究結果 (3)

| Root Type | Android | Libcore | App  | Lib  | All  | %      |
|-----------|---------|---------|------|------|------|--------|
| Runtime   | 1335    | 73      | 1843 | 690  | 3894 | 64.85% |
| Error     | 188     | 46      | 302  | 167  | 691  | 11.51% |
| Checked   | 276     | 314     | 313  | 567  | 1358 | 22.61% |
| Throwable | 0       | 0       | 2    | 0    | 2    | 0.03%  |
| Undefined | 4       | 0       | 18   | 38   | 60   | 1.00%  |
| All       | 1 803   | 433     | 2478 | 1462 | 6005 |        |

TABLE V: Types and origins of root exceptions.

全ての例外のうち約2/3はRuntimeException例外（非検査例外）

その多くがアプリケーション層から発生したもの

# 研究結果 (4)

| Origin     | stacks | signaler methods | throws clause | @throws |
|------------|--------|------------------|---------------|---------|
| Libraries  | 205    | 44               | 0             | 1       |
| Android    | 62     | 35               | 0             | 0       |
| <b>All</b> | 267    | 79               | 0             | 1       |

TABLE VI: Absence of exception interfaces on methods.

明示的に投げられたRuntimeExceptionはほんの一部  
これらのほとんどは文書化されていなかった

# 研究結果（5）

ネイティブメソッドでは、検査例外でさえも文書化されていないものがある

# 研究結果（6）

| Wrapper Type | Root Cause Type | Projects | Occurrences | Android | Java/Libcore | Lib | App |
|--------------|-----------------|----------|-------------|---------|--------------|-----|-----|
| Runtime      | Checked         | 88       | 148         | 75      | 0            | 38  | 35  |
| Runtime      | Error           | 46       | 67          | 58      | 0            | 8   | 1   |
| Checked      | Runtime         | 17       | 31          | 4       | 0            | 16  | 11  |
| Checked      | Error           | 8        | 9           | 5       | 0            | 1   | 3   |
| Error        | Checked         | 14       | 27          | 6       | 7            | 6   | 8   |
| Error        | Runtime         | 8        | 17          | 1       | 1            | 1   | 14  |

TABLE VIII: Wrappings comprising different exception types.

例外を別の例外（クロスタイプ例外）でラッピングすることがよく行われているが、これは長い例外チェーンとなって、バグを引き起こす原因になる可能性がある

# まとめ

- クロスタイプ例外による予期できないラッピングはシステムに悪影響を与える
- 文書化されていないRuntimeExceptionはAndroidプラットフォームと第三者のライブラリのどちらにも存在する
- 文書化されていない検査例外はAndroidプラットフォームから投げられる