# Characterizing the Exception Handling Code of Android Apps

Francisco Diogo Queiroz
Federal University of Rio Grande do Norte
Department of Informatics and Applied Mathematics
Natal, Brazil
diogo@ppgsc.ufrn.br

Roberta Coelho
Federal University of Rio Grande do Norte
Department of Informatics and Applied Mathematics
Natal, Brazil
roberta@dimap.ufrn.br

**Abstract— Android apps are becoming more and more popular. The number of such apps is astonishingly increasing in a daily rate, as well as the number of users affected by their crashes. Android apps as other Java apps may crash due to faults on the exception handling (EH) code (e.g. uncaught exceptions). Techniques for exception detection and handling are not an optional add-on but a fundamental part of such apps. Yet, no study has investigated the main characteristics of the EH code of mobile apps nor the developers perspective about the good and bad practices of EH in such context. This paper reports two complementary studies: one that inspected the EH code of 15 popular Android apps (which overall comprises of 3490 try-catch-finally blocks); and other study which surveyed 47 Android experts to get their opinion about the good and bad practices of EH development in Android environment. Some outcomes of the studies shows a high occurrence of exception swallowing and only few apps sending exception information to a remote server – both considered by Android experts as bad practices that negatively impact the app robustness.**

***Keywords— Exception handling; Android development; survey; exploratory study.***

## I. INTRODUÇÃO

A portabilidade e as funcionalidades oferecidas pelos dispositivos móveis atraem um número cada vez maior de usuários; e estes demandam por aplicações (apps) cada vez mais robustas. Se por um lado, as apps permitem aos celulares irem muito além das funções básicas de realizar e receber chamadas; por outro lado, elas precisam lidar com número crescente de condições excepcionais (e.g., falhas no middleware ou hardware, restrições de memória e bateria, problemas de compatibilidade [1]). Portanto, para atingir o nível de robustez exigido pelos usuários, o código dedicado ao tratamento de exceções não é uma parte opcional mas uma parte fundamental destas aplicações.

No entanto, estudos tem mostrado que o código de tratamentos de exceções está entre as partes menos compreendidas, documentadas e testadas do sistema [2, 3, 4]. Como consequência tais mecanismos podem afetar negativamente o sistema: o código relacionado ao tratamento de exceções pode introduzir falhas como exceções não capturadas (do inglês: *uncaught exceptions*) [5, 6] – que podem levar a *crashes* no sistema, tornando-o menos robusto [7].

Alguns trabalhos têm sido realizados com o objetivo de analisar características e defeitos no tratamento de exceções em aplicações Java [8, 9, 10, 11]. Alguns chegaram a analisar a frequência de exceções não capturadas lançadas por aplicações Android [12, 13]. Todavia, nenhum destes trabalhos chegou a inspecionar o código de tratamento de exceções de aplicações Android. Aplicações Android têm um contexto particular pois funcionam em um ambiente multitarefa. Além disso uma aplicação pode ter vários pontos de entrada e o ciclo de vida das aplicações é baseado no ciclo de vida dos componentes Android que exige um cuidado especial devido à complexidade [14].

Em um estudo prévio [12] foram analisados os bugs reportados à apps Android hospedadas no GitHub e GoogleCode e identificado que muitos dos defeitos e crashes reportados eram causados por defeitos no código de tratamento de exceções, mais especificamente por exceções não capturadas. Este estudo, todavia, não analisou o código de tratamento de exceções de apps Android.

Com o objetivo de melhor entender as causas para o elevado número de defeitos relacionados ao tratamento de exceções em apps Android verificados em [12], foi realizado o presente trabalho. O trabalho reportado neste artigo foi conduzido em duas etapas. Inicialmente foi realizado um estudo exploratório que analisou o código de tratamento de exceções de 15 aplicações Android populares e de código aberto disponíveis na Google Play[1]. O objetivo deste estudo foi levantar as principais características e potenciais problemas no código de tratamento de exceções destas aplicações. Foram analisados 3490 tratadores de exceções que incluíram blocos try-catch e/ou finally. Alguns resultados deste estudo foram os seguintes:

(1) o principal tratamento dado as exceções capturadas consiste em logar a exceção (44% dos tratamentos);

(2) foi observado um pequeno número de tratadores que realizavam ação de notificação ao usuário (2%);

(3) os tratadores vazios ocorreram com maior frequência quando exceções do tipo runtime eram capturadas (25%) em comparação com os tratadores vazios associados a exceções do tipo checadas (15%);

(4) 47% dos tratadores analisados eram tratadores genéricos – mais especificamente catchs com argumentos Exception (41%), RuntimeException (2%) e Throwable (4%).

Para complementar esse estudo foi realizado um survey direcionado a especialistas Android. Esses especialistas foram encontrados nas principais conferências Android de diferentes países, principais desenvolvedores das aplicações analisadas e no grupo de especialistas em Android do Google Developers Experts (GDE)[2]. Foram

[1] https://play.google.com
[2] https://developers.google.com/experts/all/technology/android

978-1-5090-5086-4/16 $31.00 © 2016 IEEE
DOI 10.1109/SBCARS.2016.25

obtidas 47 respostas para este survey que revelaram a visão dos desenvolvedores sobre tratamento de exceções.

Como resultado do survey foi elaborado um conjunto inicial de más e boas práticas relacionadas ao tratamento de exceções – segundo a opinião dos respondentes. Algumas das boas e más práticas reportadas pelos respondentes podem ser encontradas em guias que propõe como usar exceções em Java [15, 16, 17, 18]. O survey também ajudou a identificar dificuldades comuns enfrentadas durante o desenvolvimento do tratamento de exceções em apps Android.

## II. QUESTÕES DE PESQUISA

Nesta seção são apresentadas as questões de pesquisa que guiaram as duas etapas do trabalho reportado nesse artigo. A primeira questão de pesquisa está vinculada ao estudo exploratório apresentado na Seção III, as demais estão relacionadas ao survey apresentado na Seção IV. A primeira questão de pesquisa procura responder como o tratamento de exceção está sendo utilizado nas aplicações Android.

***QP 1:*** *Quais são as principais* ***características do código de tratamento de exceções*** *de aplicações Android?*

A segunda questão de pesquisa procura descobrir a visão dos desenvolvedores sobre más práticas no tratamento de exceções em aplicações Android. Esse tema foi levantado após a análise das aplicações apresentadas na Seção III.

***QP 2:*** *O que os desenvolvedores consideram como* ***má prática no tratamento de exceções*** *no desenvolvimento Android?*

Outro tema que surgiu foram os desafios e problemas encontrados pelos desenvolvedores para realizar o tratamento de exceções em aplicações Android. Procuramos descobrir também se alguma característica inerente ao ambiente Android possa causar alguma interferência negativa nesse processo.

***QP 3:*** *Quais são as* ***dificuldades enfrentadas*** *no momento de se implementar o código de* ***tratamento de exceções*** *de aplicações Android?*

Finalmente, tentamos saber a visão dos desenvolvedores para solucionar os problemas encontrados e também saber se eles utilizam algum tipo de ferramenta ou boa prática.

***QP 4:*** *Como os desenvolvedores lidam com essas dificuldades? Utilizam alguma* ***boa prática ou ferramenta****?*

## III. ESTUDO EXPLORATÓRIO: TRATADORES DE EXCEÇÕES DE APLICAÇÕES ANDROID

Nesta seção vamos descrever a metodologia utilizada para a realização do estudo exploratório que analisou o código de tratamento de exceções de 15 aplicações Android.

### *A. A seleção dasAplicações Android*

A seleção das aplicações analisadas nesse estudo foi feita manualmente, seguindo os seguintes critérios: (1) o código deveria estar disponível em algum repositório open source (e.g., GitHub, BitBucket); (2) a aplicação deveria estar na loja oficial do Android (Google Play); (3) a aplicação deveria ser popular, para isso foi definido um limite mínimo de 1 milhão de downloads. A Tabela I mostra as aplicações analisadas com a sua respectiva categoria, a quantidade de downloads, a avaliação em estrelas como também a quantidade de tratadores de exceções existentes em cada uma e a quantidade de linhas de código das aplicações.

### *B. Procedimentos da Análise*

Para analisarmos o código fonte das aplicações selecionadas utilizamos a IDE oficial do Android (Android Studio[3]), que nos permitiu realizar a construção correta das aplicações e nos forneceu as ferramentas necessárias para realizar a extração dos dados. Para procurar pelos pontos de tratamento de exceções das aplicações Android foi utilizada a seguinte expressão regular "((\s)*catch(\s)*\()|((\s)*finally(\s)*\{)", a qual foi inserida na ferramenta de busca padrão do Android Studio. Os dados das aplicações apresentados no artigo foram extraídos a partir da análise de todos os tratadores de exceções das aplicações (ver Tabela I), exceto classes de testes e código de bibliotecas.

A análise foi realizada manualmente e a categorização dos tratadores de exceções se baseou na categorização apresentada em [8], no qual é feita uma análise de tratadores de exceções de aplicações Java e .NET. Porém

TABELA I. DADOS DAS APLICAÇÕES ANALISADAS.

| # | Aplicação | Categoria | Estrelas | Nº Downloads | Tratadores Analisados | | | LoC (Java) |
|---|---|---|---|---|---|---|---|---|
| | | | | | *Catch* | *Finally* | *Total* | |
| 01 | Barcode Scanner | Compras | 4,1 | 100 ~ 500 MM | 121 | 26 | 147 | 43.031 |
| 02 | Telegram | Comunicação | 4,2 | 100 ~ 500 MM | 826 | 48 | 874 | 156.692 |
| 03 | K-9 Mail | | 4,3 | 5 ~ 10 MM | 384 | 117 | 501 | 51.576 |
| 04 | Orbot | | 4,2 | 5 ~ 10 MM | 72 | 0 | 72 | 6.489 |
| 05 | Orweb | | 4,1 | 1 ~ 5 MM | 81 | 11 | 92 | 6.560 |
| 06 | ConnectBot | | 4,6 | 1 ~ 5 MM | 83 | 15 | 98 | 19.332 |
| 07 | OpenDocument Reader | Corporativo | 4 | 1 ~ 5 MM | 37 | 9 | 46 | 3.015 |
| 08 | c:geo | Entretenimento | 4,4 | 1 ~ 5 MM | 344 | 53 | 397 | 57.388 |
| 09 | FBReader | Livros e referências | 4,6 | 10 ~ 50 MM | 348 | 41 | 389 | 76.205 |
| 10 | CoolReader | | 4,5 | 10 ~ 50 MM | 163 | 5 | 168 | 33.428 |
| 11 | Wikipedia | | 4,4 | 10 ~ 50 MM | 92 | 10 | 102 | 28.069 |
| 12 | VLC | Media & Vídeo | 4,4 | 10 ~ 50 MM | 76 | 16 | 92 | 24.684 |
| 13 | OI File Manager | Produtividade | 4,2 | 5 ~ 10 MM | 31 | 2 | 33 | 7.852 |
| 14 | Hacker's Keyboard | | 4,4 | 1 ~ 5 MM | 46 | 6 | 52 | 16.968 |
| 15 | WordPress | Social | 4,2 | 5 ~ 10 MM | 352 | 75 | 427 | 65.084 |
| | | | | **TOTAL** | **3056** | **434** | **3490** | |

[3] http://developer.android.com/intl/pt-br/tools/studio/index.html

algumas mudanças foram necessárias para refletir o contexto Android, quais sejam: (1) dois tipos foram removidos (i.e., rollback, assert); (2) o tipo alternative config. foi modificado para específico; e (3) foi criado o tipo mensagem para melhor se adaptar as características do ambiente Android. É importante ressaltar que um tratador de exceção pode conter mais de uma ação, resultando na contabilização de diferentes tipos de tratamento para o mesmo bloco de tratador. A Tabela II mostra os tipos de tratamentos utilizados nesse estudo.

### C. Resultados

Nesta seção, vamos listar os resultados que conseguimos após a análise das aplicações. Os resultados serão apresentados de acordo com a questão de pesquisa QP1.

***QP 1:*** *Quais são as principais características do código de tratamento de exceções de aplicações Android?*

Essa questão de pesquisa busca entender como que as aplicações Android estão realizando o tratamento de exceções e quais são as características do código que executa esse tratamento. A seguir serão listadas algumas características relevantes encontradas durante a análise das aplicações.

***Tipos de tratamentos.*** Após a análise dos tratadores de exceções das aplicações identificamos algumas características referentes ao código de tratamento de exceções. Utilizando os tipos apresentadas na Tabela II classificamos quais ações eram realizadas nos tratadores de exceções das aplicações Android e com que frequência essas ações eram aplicadas. A Fig.1 apresenta a porcentagem das ações extraídas de todos os tratadores (catch) de exceções analisados das aplicações Android. Podemos perceber a predominância do uso do log (44%); seguido por vazio (12%); retorna (11%); outra (11%); fecha recursos (9%) e lança com (8%). A Fig. 2 apresenta as ações realizadas nos tratadores, dividido por aplicação.

Podemos perceber que log corresponde ao maior número das ações de tratadores na maioria das aplicações analisadas, exceto em (#1, #9, #5) que tem o vazio como predominante e (#11) com lança. O uso do retorna foi relevante (27%) na aplicação (#9). É importante destacar o baixo percentual (2%) do tipo mensagem, que teve a sua maior relevância em (#13) com 19%. Das aplicações analisadas apenas (#2, #7, #11 e #15) usam ferramentas para enviar o log de falha para algum servidor. A Tabela III mostra que dentro dos blocos finally o tipo de tratamento predominante é o fecha recurso (84%), como esperado.

***Tipos de exceções capturadas.*** Durante a extração dos dados foram coletados os tipos das exceções capturadas nos tratadores de exceções (blocos try-catch). Identificamos exceções checadas e não checadas e também uma pequena quantidade de erros capturados pelos tratadores, tais como NoSuchMethodError, OutOfMemoryError, além da própria classe Error. Mesmo a exceção RuntimeException sendo do tipo não checada, ela foi contabilizada em separado por se tratar de um tipo genérico, assim como os demais supertipos de exceção. O tipo "checadas e não checadas" foi contabilizado quando os tratadores capturavam no mesmo catch exceções dos dois tipos.

Como mostra a Tabela IV, o uso do tipo Exception é predominante nas aplicações analisadas, chegando a 41% da quantidade de exceções capturadas. Com destaque para as aplicações (#2, #4, #6, #8, #10), nas quais o tipo Exception obteve o maior índice entre todos os tipos. Também podemos destacar o uso do tipo checada com 38% das exceções capturadas. A captura do tipo Throwable foi verificada nas aplicações (#2, #3, #5, #6, #7, #9, #10, #11, #12), com destaque para 15% em (#7), 11% em (#9) e 8% em (#2). Os tratadores genéricos (Exception, Throwable e RuntimeException) somam 47% dos tratadores analisados, mostrando que o tratamento genérico é comum nas aplicações analisadas.

***Tipos de exceções capturadas X Tipos de tratamentos.*** Depois de identificar os tipos de tratamentos e os tipos de exceções capturadas dentro das aplicações Android analisadas, verificamos se existe alguma relação entre essas características que nos ajude a entender melhor o tratamento de exceções no Android. Na Fig. 3 pode-se verificar um gráfico que mostra essa relação.

Nas aplicações avaliadas, uma característica observada nos tratadores associados aos tipos genéricos Exception, Throwable e RuntimeException é que a ação de log superou o valor de 50% das ações realizadas nestes tratadores, respectivamente, (68%, 51% e 68%). O alto índice no uso desse tipo juntamente com a captura desses tipos de exceções pode indicar que são poucas as alternativas de tratamento após capturá-las, restando na maioria das vezes apenas a opção de realizar o log.

Outra característica que pode ser verificada é a tendência em deixar o tratador vazio após capturar uma exceção do tipo não checada, alcançando 25%. Por outro lado, o tipo de exceção capturada com o maior índice (6%) no tipo de tratamento mensagem é exatamente o tipo não checada. O uso do tipo lança ficou mais evidenciado nos tipos de exceções checadas (15%) e não checadas (13%). No caso do tipo checadas, na maioria das vezes (63%) é gerada uma nova exceção do tipo não checada para ser

TABELA II. TIPOS DE TRATAMENTOS DE EXCEÇÕES UTILIZADAS NO ESTUDO E SUAS RESPECTIVAS DESCRIÇÕES.

| Tipos de Tratamentos | Descrição |
|---|---|
| Log | Realiza o registro de log de algum erro/exceção. |
| Específico | Realiza um tratamento específico fazendo alguma configuração alternativa. |
| Lança | Lança uma nova exceção ou relança uma já existente. |
| Vazio | O tratador está vazio, sem nenhuma instrução. |
| Continua | O tratador está dentro de um laço e força a próxima iteração do laço. |
| Retorna | O tratador força o método em execução a retornar. |
| Fecha Recursos | O código assegura que recursos abertos serão fechados. |
| Mensagem | Envia uma mensagem para o usuário. |
| Outra | Qualquer outra ação que não corresponda às anteriores. |

TABELA III. TIPOS DE TRATAMENTOS DENTRO DOS BLOCOS FINALLY.

| Tipos de Tratamentos | # | % |
|---|---|---|
| Log | 3 | 1% |
| Específico | 9 | 2% |
| Retorna | 4 | 1% |
| Fecha Recursos | 370 | 84% |
| Outra | 55 | 12% |

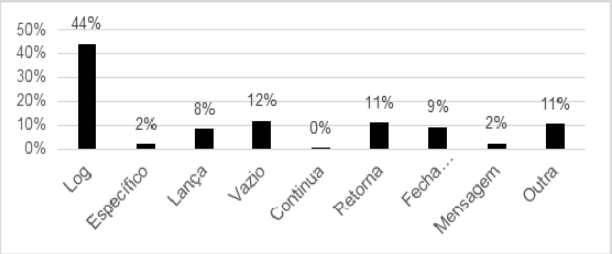


Figure 1. Quantidade de ações por tratadores de exceções das aplicações Android.

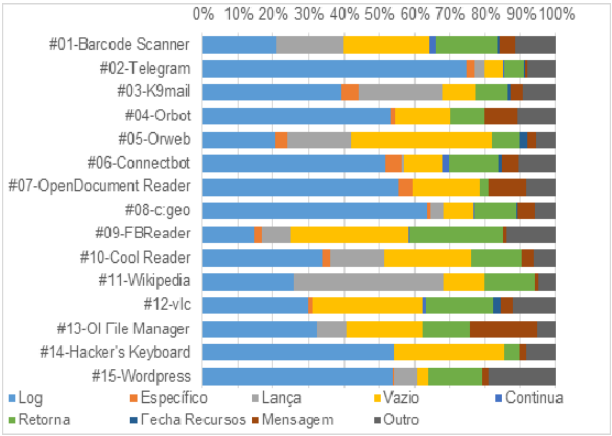


Figure 2. Distribuição dos tipos de tratamento por aplicações.

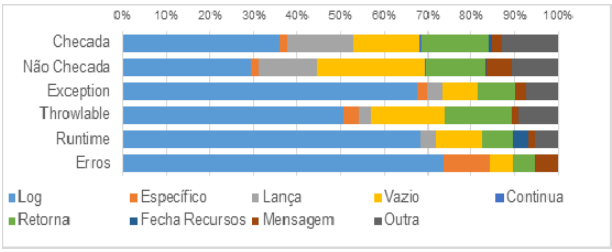


Figure 3. Relação entre os tipos de exceções capturadas nos tratadores (catch) e os tipos de tratamentos.

lançada. Já no caso do tipo não checadas, na maioria das vezes (46%) a mesma exceção capturada é relançada.

***Tipos de classes X Tipos de tratamentos.*** Outra informação extraída das aplicações foi o tipo de classe em que os tratadores se encontravam. Os tipos de classe pertinentes para esse estudo são alguns tipos específicos do Framework Android (Activity, Fragment, Service, ContentProvider, BroadcastReceiver, AsyncTask [4] ) e outros tipos de classes que têm relação com o ambiente multitarefa (Thread, Handler) e classes que implementavam a interface (Runnable). Para entender melhor como o tratamento de exceções está sendo feito dentro desses tipos classes, que fazem parte do ambiente Android, foi verificado quais tipos de tratamento eram realizados dentro de cada uma e a sua frequência.

A Fig. 4 ilustra que o tipo Activity apresenta o maior índice (14%) do tipo mensagem, indicando que o fato desse tipo de classe ser executando diretamente na UI Thread[5], facilita a notificação do usuário sobre a falha ocorrida na aplicação. Outra relação encontrada é o alto índice (38%) de lança nos tipos ContentProvider ao contrário do tipo Activity que não apresentou ocorrência desse tipo de tratamento, isso mostra que o uso do tipo lança é evitado na Activity porque o resultado seria a falha total da aplicação. Além disso, pode-se observar o uso do tipo retorna (19%) na classe AsyncTask que está atrelado à estrutura desse tipo de classe que executa uma tarefa em segundo plano e retorna um resultado.

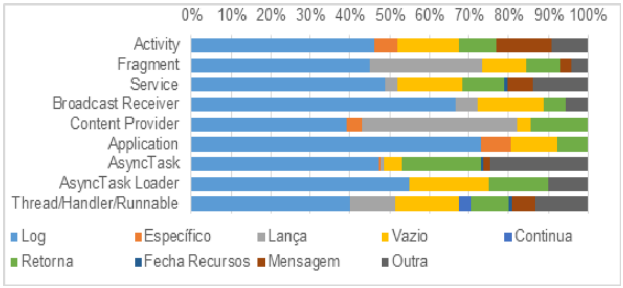


Figure 4. Relação entre os tipos de classes e os tipos de tratamentos.

TABELA IV. TIPOS DE EXCEÇÕES CAPTURADAS NOS TRATADORES (CATCH).

| Tipos de Exceções | # | % |
|---|---|---|
| Subtipos Checadas | 1152 | 38% |
| Subtipos Não Checadas | 442 | 14% |
| Supertipo Exception | 1262 | 41% |
| Supertipo Throwable | 118 | 4% |
| Supertipo RuntimeException | 49 | 2% |
| Error e subtipos | 24 | 1% |
| Checadas e Não Checadas | 9 | 0% |
| **TOTAL** | **3056** | |

## IV. SURVEY: A VISÃO DOS DESENVOLVEDORES

Esse estudo visa entender qual a visão do desenvolvedor Android sobre o tratamento de exceções em aplicações Android. Para isso, um questionário foi enviado para 1050 especialistas encontrados nas principais conferências Android (eg. droidcon, andevcon, androiddevsummit, Google I/O) que aconteceram em diferentes países.

### *A. Questionário do Estudo*

O questionário [6] é composto por seis questões de múltipla escolha e oito questões abertas. As três primeiras questões têm o objetivo de coletar informações do perfil dos especialistas, as outras onze questões são direcionadas a entender a visão do desenvolvedor em relação ao tratamento de exceções em aplicações Android.

### *B. Participantes*

Incialmente foi pensado em quem seriam os participantes do estudo exploratório. A ideia era conseguir respostas de especialistas em Android que tivessem mais experiência e a forma encontrada para alcançar esse objetivo foi procurar por palestrantes nas principais conferências sobre Android em diferentes países. Além disso, também foram contatados os principais desenvolvedores das aplicações analisadas e os especialistas em Android do GDE. O questionário online foi enviado para 1050 especialistas através de e-mail e

[4]https://developer.android.com/reference/android/os/AsyncTask.html

[5] https://developer.android.com/guide/components/processes-and-threads.html

[6] https://github.com/diogoqueiroz/android_sbcars2016

redes sociais e foram recebidas 47 respostas válidas para o questionário. O baixo percentual de respostas (4,48%) está alinhado com estudos similares como ilustrado na Seção V.

Todos os participantes do estudo têm pelo menos dois anos de experiência no desenvolvimento de aplicações Android, com 74% deles com pelo menos 4 anos de experiência, como visto na Tabela V. A maioria deles (86%) tem o desenvolvimento de aplicações Android como parte de seu trabalho. Além disso, boa parte deles (57%) desenvolvem tanto para projetos de código aberto como para projetos privados, como mostrado na Tabela VI.

### C. Procedimentos da Análise

A análise das respostas foi realizada utilizando técnicas de codificação de Grounded Theory [19] para extrair as informações necessárias das respostas do questionário e conseguir responder as questões de pesquisas. A técnica de codificação utilizada nesse estudo consiste em duas fases: (1) uma codificação inicial - que implica em uma leitura minuciosa dos dados e (2) uma codificação específica para identificar e desenvolver os temas mais relevantes [19] nos dados analisados. O processo de codificação foi realizado por dois codificadores que concordaram com os temas que emergiram durante a codificação. É importante ressaltar que as respostas dos desenvolvedores, mostradas nesse artigo, foram traduzidas do inglês para o português.

### D. Resultados

Para ilustrar os resultados encontrados durante a análise, utilizamos algumas citações provenientes do survey. As citações seguem o padrão [D#] para mantermos o anonimato dos desenvolvedores participantes e conseguirmos rastreá-los no nosso repositório. Por exemplo, o código [D1] corresponde a uma resposta fornecida pelo desenvolvedor 1. A seguir os resultados serão separados por questões de pesquisa.

***QP 2:*** *O que os desenvolvedores consideram como má prática no tratamento de exceções no desenvolvimento Android?*

Os desenvolvedores foram questionados sobre o que eles consideravam más práticas no tratamento de exceções em aplicações Android. A Tabela VII lista as más práticas que emergiram durante análise das respostas dos desenvolvedores. Em seguida, é realizado um descritivo sobre as más práticas mais relevantes.

TABELA V. EXPERIÊNCIA DOS DESENVOLVEDORES NO DESENVOLVIMENTO DE APLICAÇÕES ANDROID.

| Experiência em Android | % |
|---|---|
| < 2 anos | 0% |
| >= 2 anos e < 4 anos | 26% |
| >= 4 anos e < 6 anos | 40% |
| >= 6 anos | 34% |

TABELA VI. TIPOS DE PROJETOS QUE OS DESENVOLVEDORES ANDROID TRABALHAM.

| Tipo do Projeto | % |
|---|---|
| Código aberto | 14% |
| Código fechado | 24% |
| Ambos | 57% |
| Outro | 5% |

TABELA VII. MÁS PRÁTICAS DE TRATAMENTO DE EXCEÇÕES EM APPS ANDROID MAIS CITADAS PELOS DESENVOLVEDORES (42 RESPOSTAS).

| Más Práticas | # | % |
|---|---|---|
| Silenciar exceções | 16 | 38% |
| Capturar tipos genéricos | 12 | 29% |
| Não fornecer uma boa mensagem de erro ao usuário | 5 | 12% |
| Não enviar o log do erro para um servidor | 4 | 10% |
| Uso de try/catch para simular estabilidade da app | 4 | 10% |
| Usar exceção para o fluxo normal | 2 | 5% |
| Permitir exceções não capturadas | 2 | 5% |
| Grandes blocos try/catch | 2 | 5% |
| Usar printstacktrace | 2 | 5% |
| Logar todos os erros | 2 | 5% |

***Silenciar exceções.*** *"Try/catch sem nada dentro do bloco catch"* [D10]; *"Ignorar exceção quando o usuário espera um retorno da ação realizada"* [D42].

O ato de silenciar ou ignorar uma exceção foi a má prática mais citada (38%) pelos desenvolvedores que responderam ao questionário. Eles consideram que essa prática pode gerar inconsistências nas aplicações, prejudicando assim o funcionamento das mesmas. Além disso, acrescentam que ao deixar o bloco catch sem nenhuma instrução (vazio), o desenvolvedor não apenas está colocando em risco o funcionamento correto da aplicação como também está abdicando da possibilidade de saber se está acontecendo algo de errado com a aplicação ao não enviar um log com o erro para algum servidor.

*Silenciar ou ignorar uma exceção foi a má prática mais citada pelos desenvolvedores (38%); e foi identificada em 12% dos tratadores analisados.*

***Capturar tipos genéricos.*** *"Você deve sempre tentar capturar a exceção mais precisa possível, em vez de capturar uma das exceções genéricas. Se você captura uma exceção genérica, você pode estar capturando algo que você não pretendia (e consequentemente algo que talvez você não possa se recuperar)"* [D17].

Os desenvolvedores acreditam que capturar exceções do tipo genérico (ex.: Exception, RuntimeException, Throwable) pode gerar inconsistências na aplicação. Uma vez que um desenvolvedor cria um catch com uma exceção genérica (ex. Exception), esse ponto pode capturar várias outras exceções que o desenvolvedor não pensou anteriormente. Portanto, qualquer exceção que aconteça no escopo desse catch será capturada e não necessariamente precisará do mesmo tipo de tratamento que foi previamente desenvolvido. Outra questão levantada pelos desenvolvedores é utilizar essa má prática combinada com a má prática de silenciar exceções, em que um desenvolvedor utiliza um catch genérico para capturar todas as exceções que possam ocorrer naquele escopo e em seguida simplesmente ignorá-las.

Embora boa parte dos desenvolvedores (29%) tenham considerado capturar exceções genéricas como uma má prática, alguns disseram que utilizam essa prática em algumas situações – *"Eu não vejo nada de errado em capturar exceções genéricas se o código de recuperação do erro é o mesmo para todos os casos possíveis. Entretanto, capturar Throwable parece ser muito genérico [...]"* [D22].

*Os tratadores genéricos somaram 47% dos tratadores analisados, mostrando que o tratamento genérico foi comum nas aplicações analisadas.*

***Não fornecer uma boa mensagem de erro ao usuário.*** *"Capturar exceções sem prover uma mensagem sobre o problema para o usuário, de forma que a experiência do usuário seja quase tão negativa quanto a falha da aplicação"* [D1].

Existe uma certa preocupação por parte dos desenvolvedores Android com a experiência que o usuário tem em relação as suas aplicações. A concorrência entre as aplicações na loja oficial do Android é grande, visto que existem várias aplicações com o mesmo propósito e qualquer problema que a aplicação tenha pode fazer com que os usuários desistam dela e escolham uma outra opção. Dessa forma, o sucesso da aplicação fica comprometido se o usuário recebe mensagens de erro muito vagas como citado por [D4].

Outra prática desencorajada pelos desenvolvedores é mostrar detalhes internos da exceção para o usuário, algo que só deve acontecer no ambiente de desenvolvimento ou em um eventual log do erro.

*Nos tratadores das aplicações analisadas essa ação foi pouco utilizada, indicando que os usuários não recebem mensagens específicas sobre a maioria das falhas que acontecem nas aplicações.*

***Não enviar o log do erro para um servidor.*** *"Registrar o log localmente mas não enviar para algum tipo de serviço"* [D4].

As aplicações Android executam nos dispositivos móveis dos usuários, portanto ao realizar normalmente o log de uma exceção o desenvolvedor estará registrando o log localmente no dispositivo do usuário. Desse modo, o desenvolvedor acaba não sabendo nada sobre os erros que estão acontecendo na sua aplicação. Mesmo realizando vários testes durante a fase de desenvolvimento, tem sempre a possibilidade de existirem erros não detectados, pois são vários os dispositivos e cada um com suas peculiaridades e que podem rodar diferentes versões do sistema operacional Android. Portanto, é importante sempre enviar o log dos erros para algum servidor para que a equipe de desenvolvimento fique ciente dos erros que estão acontecendo na aplicação.

Esta resposta motivou uma nova análise do código das apps, na qual procuramos pelos dois tipos de ferramentas mencionadas pelos desenvolvedores. Observamos que 20% utilizava uma das ferramentas específicas de report de falhas e 27% sobrescrevia a UncaughtExceptionHandler[7] para armazenar o erro em um arquivo que pode ser lido posteriormente com autorização do usuário ou solicitando ao usuário que envie um e-mail com o log do erro ocorrido, como mostrado na Tabela VIII.

*Aproximadamente 47% das apps enviavam informações sobre exceções uncaught para um servidor remoto.*

[7]https://developer.android.com/reference/java/lang/Thread.UncaughtExceptionHandler.html

TABELA VIII. ESTRATÉGIAS PARA ENVIAR O LOG PARA O DESENVOLVEDOR

| Soluções para envio de log | # Apps | % Apps |
|---|---|---|
| Usa ferramentas de *report de falhas* | 3 | 20% |
| Sobrescreve UncaughtExceptionHandler | 4 | 27% |
| **TOTAL** | **7** | **47%** |

***Uso excessivo de try-catches.*** *"Abusar de try/catchs levando a circunstâncias em que o erro que deveria ser consertado é na verdade silenciado e nunca terá a atenção do desenvolvedor."* [D34].

Alguns desenvolvedores falaram sobre os efeitos negativos do uso excessivo e não sistemático de blocos try-catch para "simular" estabilidade nas apps. Se uma exceção ocorreu e se não for dado um tratamento adequado o sistema entrará em um estado inconsistente. Assim, tentar "esconder" a exceção do usuário pode ser tão prejudicial quanto uma tela de crash com o agravante de não prover informações para a correção do problema reportado pela exceção.

*O uso abusivo de try-catches pode silenciar erros que nunca serão corrigidos pois não chegarão ao conhecimento dos desenvolvedores ou testadores. Em um cenário de crash pelo menos o bug é percebido e pode ser corrigido.*

***QP 3:*** *Quais são as dificuldades enfrentadas no momento de se implementar o código de tratamento de exceções de aplicações Android?*

Essa questão de pesquisa procura entender se existem aspectos específicos da Plataforma Android que possam dificultar o tratamento de exceções durante o desenvolvimento das aplicações.

Primeiramente, os desenvolvedores foram questionados se acreditavam na existência de alguma característica inerente a plataforma Android que pudesse dificultar o tratamento de exceções nas aplicações. A maior parte dos desenvolvedores informaram que sim (68%), que existe pelo menos uma característica da plataforma Android que dificulta o tratamento de exceções. Outros desenvolvedores responderam não (26%) e o restante (6%) informaram que não sabiam responder.

Em seguida, os desenvolvedores que responderam positivamente à questão anterior foram perguntados sobre quais são as características inerentes à Plataforma Android que podem causar problemas ao tratamento de exceções. A questão listava previamente algumas características para múltipla seleção que também incluía a opção para acrescentar outras características não listadas.

Como pode ser visto na Tabela IX, considerando que os desenvolvedores poderiam escolher mais de uma, a característica mais selecionada pelos desenvolvedores como causadora de problemas para o tratamento de exceções foi o ciclo de vida dos componentes Android (71%), seguida pelo ambiente multitarefa do Android (58%), vários pontos de entrada para uma aplicação (32%) e outras (32%). Das características citadas além das opções fornecidas previamente, podemos destacar: mudança de contexto e rotação; problemas específicos de dispositivos; a dependência do sistema e de bibliotecas; e a alocação de memória para imagens.

TABELA IX. CARACTERÍSTICAS QUE PODEM PROVOCAR PROBLEMAS NO TRATAMENTO DE EXCEÇÕES.

| Característica | # | % |
|---|---|---|
| Ciclo de vida dos componentes Android | 22 | 71% |
| Ambiente multitarefa do Android | 18 | 58% |
| Vários pontos de entrada para uma aplicação | 10 | 32% |
| Outras | 10 | 32% |

Entre as dificuldades citadas pelos desenvolvedores, as mais relevantes foram: a dificuldade em lidar com o ciclo de vida da Activity/Fragment; comunicação entre a tarefa em segundo plano e a tarefa principal (UI Thread) e múltiplos pontos de entrada.

***Ciclo de vida da Activity/Fragment.*** *"O ciclo de vida provoca uma série de problemas, especialmente para fragmentos dentro de Activity. Isso adiciona complexidade ao ciclo de vida do objeto o que significa que algo pode ser nulo ou em um estado diferente do que deveria ser quando você for usar"* [D17].

Muitos dos desenvolvedores relataram problemas com o ciclo de vida dos componentes da Plataforma Android, mas não deixaram claro como exatamente essa característica poderia causar problemas diretamente ao tratamento de exceções. Os relatos dos desenvolvedores transmitem a mensagem de que a complexidade dos ciclos de vida, principalmente do fragmento e da atividade, pode causar problemas de uma forma geral no desenvolvimento de aplicações Android. [D7] nos disse que o uso de fragmento é uma fonte constante de exceções não esperadas, cuja a principal razão é a complexidade da API do Fragment. Outros desenvolvedores informaram que o uso de fragmentos nas aplicações Android torna difícil a manipulação das mudanças de configurações, pois o fragmento geralmente adere ao ciclo de vida da aplicação. Também foi mencionado por outros três desenvolvedores que o uso de fragmentos aninhados deve ser evitado, pois esse procedimento aumenta ainda mais a complexidade e consequentemente os problemas ao usar fragmentos.

***Comunicação entre tarefas.*** *"É frequente a necessidade de remover algumas tarefas da UI Thread. Se essas tarefas falham, pode ser necessário comunicar a falha ao usuário através da UI Thread. Isso significa capturar uma exceção em uma tarefa em segundo plano e relançar essa exceção (ou algo similar) de volta para a UI Thread. Isso pode ser complicado. Além disso, se uma falha ocorre no momento em que a aplicação tem baixa prioridade e, por conseguinte, é finalizada, é possível perder completamente os dados sobre a falha"* [D20].

A comunicação entre as tarefas de segundo plano e a tarefa principal gera dificuldades no tratamento de exceções, visto que qualquer falha que ocorra em uma tarefa de segundo plano terá que ser repassada de alguma forma para a UI Thread para que o usuário receba uma mensagem adequada.

O desenvolvedor [D25] informou que quanto mais interação com a UI na aplicação maiores as chances de ter problemas com o tratamento de exceções. [D1] acrescenta que quando uma tarefa assíncrona (em segundo plano) guarda uma referência a um objeto que já tenha sido destruído, pode resultar em erros. Esses problemas causados por referências a objetos já destruídos acontecem pela forma como o SO Android lida com as aplicações em execução [14].

***Múltiplos pontos de entrada.*** *"Os vários pontos de entrada é uma enorme fonte de potenciais bugs, porque isso significa que sua aplicação pode estar em vários estados possíveis e você tem que se planejar para cada um desses estados"* [D37].

Os desenvolvedores selecionaram os múltiplos pontos de entrada como sendo um problema, mas da mesma forma que no ciclo de vida dos componentes não deixaram claro como esse problema atinge diretamente o tratamento de exceções. Uma aplicação Android pode ter múltiplos pontos de entrada, o que significa que ela pode ser inicializada em vários pontos, de modo que o mais comum é a inicialização através de uma Activity principal que é executada quando o usuário clica no ícone da aplicação. Além disso, uma aplicação pode ser iniciada: através de um serviço requisitado por outro aplicativo ou pelo sistema operacional; por um BroadcastReceiver recebendo um Intent implícito que poderá abrir uma Activity diferente da principal; etc. Todo esse contexto gera complexidade na manipulação do estado da aplicação, sendo possíveis fontes de bugs e aumentando a complexidade no tratamento dos mesmos.

***QP 4:*** *Como os desenvolvedores lidam com essas dificuldades? Utilizam alguma boa prática ou ferramenta?*

Nessa questão de pesquisa, procuramos saber quais as medidas tomadas pelos desenvolvedores para sanar os problemas levantados na QP 3, como também quais as boas práticas utilizadas por eles no desenvolvimento do código de tratamento de exceções de aplicações Android. Os temas emergidos durante a análise serão categorizados em soluções para os problemas mencionados anteriormente e boas práticas. A Tabela X lista as boas práticas que emergiram após a análise das respostas.

TABELA X. BOAS PRÁTICAS DE TRATAMENTO DE EXCEÇÕES EM APPS ANDROID MAIS CITADAS PELOS DESENVOLVEDORES (43 RESPOSTAS).

| Boas Práticas | # | % |
|---|---|---|
| Usar ferramentas de report de falhas | 10 | 23% |
| Notificar o usuário | 7 | 16% |
| Falhar rapidamente | 7 | 14% |
| Capturar apenas se puder se recuperar | 7 | 14% |
| Não silenciar as exceções | 4 | 9% |
| Logar todas as exceções | 4 | 7% |
| Tratamento específico | 3 | 7% |
| Documentar as exceções | 3 | 7% |
| Fechar recursos nos blocos finally | 2 | 5% |
| Usar biblioteca específica de log | 2 | 5% |
| Não adiar o tratamento | 2 | 5% |
| Não usar printstacktrace | 2 | 5% |

***Solução: Evitar o uso de Fragment.*** *"Eu lido com isso[complexidade do ciclo de vida] principalmente evitando fragmentos aninhados tanto quanto possível e usando somente a versão do Fragment fornecido pela biblioteca de suporte, que provê uma implementação de API única através de várias versões da plataforma"* [D7].

Como visto anteriormente, os desenvolvedores não direcionaram problemas específicos para o tratamento de exceções referentes ao ciclo de vida dos componentes do Framework Android mas mostraram que de uma forma geral a complexidade do uso de fragmentos tem causado problemas e provocado o surgimento de soluções alternativas.

Três desenvolvedores sugeriram evitar o uso de fragmentos nas aplicações Android, pois acreditam que o

seu uso traz mais problemas que benefícios. O desenvolvedor [D34] disse que para suprir essa lacuna ele utilizada a combinação de duas bibliotecas (mortar[8] e flow[9]) que permite o gerenciamento de múltiplas telas utilizando apenas uma única Activity como base para a exibição das mesmas. Dessa forma, ele informa que consegue se livrar do uso dos fragmentos em suas aplicações. Outra ferramenta citada por [D12] para auxiliar no uso de fragmentos aninhados é a Otto[10] que ajuda na comunicação entre os fragmentos e as atividades.

***Solução: Gerenciar ambiente multitarefas.*** *"Para multitarefa temos uma biblioteca própria que nos ajuda. Ela sincroniza com a UI Thread, então as rotações se tornam simples. Nós também realizamos revisões de código para evitar que o código da UI Thread e das tarefas de segundo plano utilizem o mesmo arquivo de forma simultânea"* [D4].

O desenvolvedor Android não pode fugir do ambiente multitarefa inerente ao sistema operacional, portanto, para reduzir os problemas que essa característica pode gerar eles têm que criar soluções que os ajudem a superá-las. Soluções próprias como a criação de uma biblioteca específica para auxiliar na comunicação entres as várias tarefas assíncronas que possam estar executando na aplicação com a tarefa principal é uma forma de evitar tais problemas. O uso da biblioteca RxJava[11] foi citado por dois desenvolvedores, que informaram que a ferramenta substitui eficazmente a forma padrão do Android de interagir com o ambiente multitarefa.Além disso, eles disseram que o uso da RxJava traz benefícios ao tratamento de erros porque ela provê um sistema que utiliza o método de callback onError() para escutar os erros/exceções que possam acontecer na aplicação.

***Boa prática: Uso de ferramentas de report de falhas.*** *"Elas [ferramentas] nos dão pistas sobre como a aplicação está sendo executada. Sem isso não tem como sabermos o que está acontecendo com a aplicação nos dispositivos dos usuários finais"* [D33].

A maioria dos desenvolvedores (93%) que responderam ao questionário utilizam esse tipo de ferramenta em suas aplicações. Eles acreditam que esse tipo de ferramenta tem um impacto positivo no processo de criação de uma aplicação mais robusta e confiável. Aproximadamente 24% dos desenvolvedores disseram que esse tipo de ferramenta fornece um bom retorno para a equipe de desenvolvimento, pois além de enviar o log das exceções para um servidor, elas também fornecem funcionalidades internas para analisar os tipos e frequência das falhas que ocorrem na aplicação. Essas funcionalidades permitem que o desenvolvedor faça um estudo dos pontos mais críticos, priorizando assim o conserto das falhas consideradas mais urgentes. Nas respostas dos desenvolvedores foi citado explicitamente o uso das ferramentas Crashlytics (Fabric[12]), Crittercism (Apteligent[13]) e ACRA[14].

Embora a maioria tenha dito que usam esse tipo de ferramentas, alguns desenvolvedores chegaram a afirmar que não as utilizam – *"Adiciona complexidade, requisitos extra de memória, necessidade de configuração e suporte, considerações de privacidade [...]"* [D27]. Outro desenvolvedor disse que a ferramenta padrão fornecida pelo Google é suficiente, pois suas aplicações são pequenas e não muito populares e, portanto, não precisam de monitoramento contínuo.

***Boa prática: Notificar o usuário.*** *"[...] mostre o problema para o usuário decidir o que fazer quando você não puder fazer nada automaticamente"* [D6]

A prática de notificar o usuário quando uma falha ocorre na aplicação foi mencionada explicitamente por sete desenvolvedores. Eles falaram que se a falha é crítica, então notifique o usuário e talvez permita que ele tente realizar a ação novamente. Alguns deles reforçam que em casos de falhas mais simples que não interfiram no fluxo da aplicação não é necessário interromper o usuário com um alerta – *"[...]não interrompa o usuário (por exemplo, caso não carregue a imagem do avatar de algum contato)"* [D14]. E acrescentam que sempre que for notificar o usuário sobre uma falha utilize mensagens adequadas – *"Mostrar uma mensagem educada, legível e traduzida e nunca mostrar a classe da exceção ou o stacktrace"* [D22].

***Boa prática: Falhar rapidamente.*** *"Falhe o quanto antes possível para prevenir estados inesperados no futuro"* [D38].

Sete desenvolvedores destacaram que em casos de exceções inesperadas, o melhor a ser feito é falhar rapidamente, ou seja, deixar que a exceção finalize a aplicação. Alguns desenvolvedores ressaltam que exceções do tipo RuntimeException (NullPointerException, IllegallStateException, etc.) que representam erros de programação devem sempre provocar a falha na aplicação. O desenvolvedor [D9] diz que é melhor acontecer a falha da aplicação do que deixá-la em um estado inconsistente – "Proceder com um estado incerto pode tornar ainda mais difícil de diagnosticar erros em produção, e frequentemente sua aplicação será avaliada pelos usuários como sendo um software de baixa qualidade".

Alguns desenvolvedores chegaram a citar que no Android a falha geral da aplicação é uma experiência bastante negativa para o usuário, por exemplo, *"Em Android, a falha da aplicação é uma experiência particularmente negativa para o usuário, e geralmente é preferível prover ao usuário uma aplicação com uma funcionalidade ou UI levemente degradada do que falhar completamente"* [D1].

***Boa prática: Capturar apenas se puder tratar.*** *"Eu capturo uma exceção se eu entendo as circunstâncias do seu lançamento, se eu posso lidar com ela e se eu posso recuperar-se de forma segura para que a aplicação mantenha seu funcionamento correto"* [D17].

Existe uma preocupação dos desenvolvedores em capturar apenas as exceções que eles entendem e que possam tratá-las de forma razoável, sete desenvolvedores relataram essa prática. Eles entendem que se não existe um plano claro para realizar a recuperação não se deve capturar a exceção. Alguns desenvolvedores listaram práticas mais específicas, como – *"[...] eu costumo capturar exceções em lugares que não afetam muito o*

[8] https://github.com/square/mortar
[9] https://github.com/square/flow
[10] http://square.github.io/otto/
[11] https://github.com/ReactiveX/RxJava
[12] https://try.crashlytics.com/
[13] https://www.apteligent.com/product/crash-reporting/
[14] http://www.acra.ch/

*fluxo da aplicação ou se a situação é recuperável [...] "* [D18].

***Boa prática: Documentar exceções.*** *"Documente exceções recuperáveis como parte da API"* [D20].

Os desenvolvedores acreditam que a documentação das exceções pode não apenas reduzir o número de falhas da aplicação como também pode auxiliar no tratamento das mesmas, caso ocorram. Uma vez que o desenvolvedor sabe que em um determinado ponto da aplicação são lançadas exceções e tem o conhecimento de quais são essas exceções, ele terá muito mais facilidade para escolher o tipo adequado de tratamento que deverá ser realizado para a situação em questão. A documentação é importante principalmente no caso das exceções do tipo não checadas que não tem obrigatoriedade de serem tratadas em Java – *"[...] documente todas as exceções, principalmente as do tipo RuntimeException [...]"* [D36].

## V. TRABALHOS RELACIONADOS

Os trabalhos relacionados a este estudo podem ser divididos nas seguintes categorias: (i) estudos empíricos que analisaram o código e/ou os bugs relacionados ao tratamento de exceções de sistemas Java [8, 9, 10, 11]; (ii) estudos que analisaram os bugs relacionados ao tratamento de exceções de apps Android [12, 13]; e (iii) *surveys* aplicados a desenvolvedores Android [20, 21, 22].

O trabalho de Cabral e Marques [8] levantou características do tratamento de exceções em aplicações Java e .NET. Este estudo analisou 16 aplicações Java e 16 .NET, classificadas nos seguintes grupos: bibliotecas, *stand-alone, aplicações servidoras e aplicações Web.* Foram examinados manualmente os blocos try e seus correspondentes tratadores. Este trabalho não avaliou o tratamento de exceções no contexto de aplicações Android. Todavia, podemos fazer algumas comparações com relação aos resultados encontrados nos dois trabalhos. Na maioria dos grupos de aplicações Java analisados por Cabral e Marques [8] (com exceção das bibliotecas) o tipo de tratamento mais comum foi a ação de log (28% em média). No nosso estudo, a ação de log também foi a mais comum no tratamento de exceções das apps Android (44%). Além disso, o percentual de tratadores vazios que foi superior a 10% em todos os grupos de aplicações Java analisadas em [8] tiveram uma ocorrência similar nas apps Android analisadas no nosso trabalho (12%).

O trabalho realizado por Cacho et al [11] avaliou como mudanças no código *normal* e no código de tratamento de exceções estavam relacionados a defeitos no código de tratamento de exceções de programas C# e Java. O estudo conduzido por Barbosa et al [9] criou categorias de bugs de tratamento de exceções baseados na análise do repositório de bugs e do código associado a estes bugs de duas aplicações: o Apache Tomcat e o framework Hadoop. Um dos bugs identificados por este trabalho foram os tratadores vazios, também identificados no nosso trabalho, porém este trabalho não analisou os demais tipos de tratamento.

O trabalho apresentado por Ebert et al [10] é similar ao trabalho conduzido por Barbosa et al [9]. Ele define categorias de bugs de tratamentos de exceções. Porém, além de analisar 220 bugs de tratamento de exceções dos repositórios das aplicações Eclipse e Tomcat, o estudo também aplicou um *survey* para 154 desenvolvedores. O *survey* focou em como políticas e padrões para implementação de tratamento de erros estariam relacionadas a cultura das organizações e na documentação de tratamento de exceções na fase de projeto – tendo, portanto, um foco diferente do nosso *survey*.

Coelho et al [12] realizaram um estudo que analisou as stack traces extraídas de issues de projetos Android hospedados no GitHub (482 projetos) e Google Code (157 projetos). O objetivo deste estudo foi investigar se a partir da análise de stack traces era possível identificar deficiências no tratamento de exceções destas apps. Este trabalho não chegou a analisar o código fonte das apps e o comportamento dos tratadores como realizado no nosso trabalho. Kechagia e Spinellis [13] examinaram os stack traces enviados por aplicações Android para um serviço de gerenciamento de falhas (i.e., BugSense). Eles verificaram que 19% dos stack traces eram causados por exceções não checadas e não documentadas lançadas por métodos definidos na API do Android.

Kochhar et al [20] aplicaram um questionário para descobrir as ferramentas comumente utilizadas para o teste de aplicações mobile assim como os problemas enfrentados pelos desenvolvedores ao testarem estas aplicações. O questionário foi enviado para 3.905 e-mails de desenvolvedores Android e obteve 83 respostas (taxa de resposta de 2.13%). No trabalho de Joorabchi et al [21] foi aplicado um questionário com o objetivo de melhor entender os principais desafios enfrentados pelos desenvolvedores ao construir apps para diferentes tipos de dispositivos. Eles entrevistaram 12 desenvolvedores mobile sênior e realizaram um questionário com 188 desenvolvedores. O questionário foi compartilhado em grupos de e-mail e mídias sociais. Por fim, o estudo apresentado por Linares-Vásquez et al [22] aplicou um questionário a desenvolvedores Android para identificar as práticas utilizadas para detectar gargalos de desempenho em apps - 24.340 e-mails foram enviados e 628 respostas foram recebidas - taxa de resposta de 2.6%.

Uma característica comum destes *surveys* são as baixas taxas de resposta – entre 2% e 4%. Nenhum desses questionários, porém, investigou as práticas utilizadas para lidar com o tratamento de exceções neste contexto. No nosso estudo realizamos um questionário exploratório com objetivo de saber quais as práticas e quais os problemas, na visão dos desenvolvedores, no tratamento de exceções em aplicações Android.

## VI. LIMITAÇÕES

***Análise das Aplicações.*** A escolha das aplicações foi definida por critérios que buscavam aplicações bem classificadas e que tivessem o código-fonte aberto. Embora esses critérios tenham eliminado aplicações pouco relevantes no cenário do Android, eles não garantem que as aplicações selecionadas utilizam as melhores práticas existentes no tratamento de exceções de aplicações Android. O grupo de 15 aplicações analisadas, apesar de pertencer a categorias diferentes, não é estatisticamente representativo para todas as aplicações Android. Porém, os resultados dão indicativos para as características do código de tratamento de exceções de apps Android. Outra limitação é que mesmo seguindo o modelo proposto em [8] para definir os tipos de tratamento de exceções em aplicações Android, a classificação em alguns casos pode

ser subjetiva e no estudo em questão a classificação e análise foram realizadas por uma única pessoa. Estas ameaças são similares as de outros trabalhos que analisaram o código de tratamento de exceções de sistemas Java [12][23].

***Survey.*** Devido à natureza exploratória desse estudo que tem como objetivo entender a visão dos desenvolvedores sobre o tratamento de exceções em aplicações Android, nós escolhemos técnicas de Grounded Theory. Os resultados encontrados podem não ser aplicados a todos os desenvolvedores, uma vez que existem outras populações que podem adicionar novas perspectivas. A população dos participantes ficou restrita a especialistas palestrantes em conferências Android de diferentes países e a um grupo de especialistas Android determinado pelo Google. A busca por essas conferências aconteceu através de pesquisas na internet, o que não garante que todas as conferências Android importantes tenham sido selecionadas. Embora não possamos generalizar os temas que emergiram durante a análise dos dados, eles nos fornecem uma visão inicial das dificuldades dos desenvolvedores e de como eles lidam com elas durante o desenvolvimento do tratamento de exceções de aplicações Android.

## VII. CONCLUSÕES

Neste trabalho apresentamos dois estudos complementares realizados com objetivo de melhor entender como o tratamento de exceções em aplicações Android vem sendo implementado e qual a opinião dos desenvolvedores sobre boas e más práticas neste contexto. O primeiro estudo analisou o código dos tratadores de exceções de quinze aplicações Android. Durante esse estudo foi possível identificar os principais tipos de tratamentos de exceções e a frequência com que esses tipos ocorreram nas aplicações. Além disso, foi possível relacioná-las com os tipos de exceções capturadas pelos tratadores e os tipos de classes em que esses tratadores se encontravam.

No segundo estudo, foi realizado um survey com objetivo de entender a perspectiva dos desenvolvedores sobre o tratamento de exceções em aplicações Android. Esse estudo foi realizado através da aplicação de um questionário para 47 especialistas Android. Com os resultados foi possível definir, na visão dos desenvolvedores, um conjunto de más e boas práticas relacionadas ao tratamento de exceções como também as dificuldades enfrentadas ao lidar com exceções em aplicações Android.

Esse trabalho mapeou de forma inicial o tratamento de exceções de aplicações Android; os resultados apontam para a necessidade de guias/padrões que auxiliem os desenvolvedores a lidar com situações excepcionais comuns ao ambiente de desenvolvimento de aplicações Android.

## REFERÊNCIAS


[1] T. McDonnell, B. Ray and M. Kim, “An empirical study of api stability and adoption in the android ecosystem”. In: Proc. International Conference on Software Maintenance (ICSM), 2013, pp 70–79.

[2] R. Miller and A. Tripathi, “Issues with exception handling in object-oriented systems”. In European Conference on Object-Oriented Programming, Springer Berlin Heidelberg, June 1997, pp. 85-103.

[3] M. P. Robillard and G. C. Murphy, “Designing robust Java programs with exceptions”. In ACM SIGSOFT Software Engineering Notes, ACM , Vol. 25, No. 6, Nov. 2000, pp.. 2-10,

[4] H. Shah, C. Gorg and M. J. Harrold, “Understanding exception handling: Viewpoints of novices and experts”. IEEE Transactions on Software Engineering, 36(2), 2010, pp. 150-161.

[5] J. W. Jo, B. M. Chang, K. Yi and K. M. Choe, “An uncaught exception analysis for Java”. Journal of systems and software, 72(1), 2004, 59-69.

[6] P. Zhang and S. Elbaum, “Amplifying tests to validate exception handling code”. In Proc. of the 34th International Conference on Software Engineering, IEEE Press, June 2012, pp. 595-605.

[7] R. Coelho, A. von Staa, U. Kulesza, A. Rashid and C. Lucena “Unveiling and taming liabilities of aspects in the presence of exceptions: A static analysis based approach”. Information Sciences, 181(13), 2011, pp. 2700-2720.

[8] B. Cabral and P. Marques, “Exception handling: A field study in Java and .Net”. In European Conference on Object-Oriented Programming, Springer Berlin Heidelberg, July 2007, pp. 151-175.

[9] E. A. Barbosa, A. Garcia and S. D. J. Barbosa, “Categorizing faults in exception handling: A study of open source projects”. In Brazilian Symposium Software Engineering, 2014, pp. 11-20.

[10] F. Ebert, F. Castor and A. Serebrenik, “An exploratory study on exception handling bugs in Java programs”. Journal of Systems and Software, 106, 2015, pp. 82-101.

[11] N. Cacho, E. A. Barbosa, J. Araujo, F. Pranto, A. F. Garcia, T. Cesar, E. Soares, A. Cassio, T. Filipe and I. Garcia, “How Does Exception Handling Behavior Evolve? An Exploratory Study in Java and C# Applications”. In ICSME, 2014, pp. 31-40.

[12] R. Coelho, L. Almeida, G. Gousios and A. van Deursen, “Unveiling exception handling bug hazards in Android based on GitHub and Google code issues”. In Proc. of the 12th Working Conference on Mining Software Repositories, 2015, pp. 134-145.

[13] M. Kechagia and D. Spinellis, “Undocumented and unchecked: exceptions that spell trouble”. In Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 312-315.

[14] A. Goransson, Efficient Android Threading: Asynchronous Processing Techniques for Android Applications. "O'Reilly Media, Inc.", 2014, pp. 1-11.

[15] D. Mandrioli and B. Meyer, Advances in object-oriented software engineering. Prentice-Hall, Inc, 1992.

[16] J. Gosling, The Java language specification. Addison-Wesley Professional, 2000.

[17] R. J. Wirfs-Brock, “Toward exception-handling best practices and patterns”. IEEE software, 23(5), 2006, pp. 11-13.

[18] J. Bloch, Effective Java. Pearson Education India, 2008.

[19] K. Charmaz, “Constructing grounded theory: A practical guide through qualitative research”. SagePublications Ltd, London, 2006.

[20] P. S. Kochhar, F. Thung, N. Nagappan, T. Zimmermann and D. Lo, “Understanding the test automation culture of app developers”. In Software Testing, Verification and Validation (ICST), 2015 IEEE 8th International Conference on, IEEE, April 2015, pp. 1-10.

[21] M. E. Joorabchi, A. Mesbah and P. Kruchten, “Real challenges in mobile app development. In Empirical Software Engineering and Measurement”, 2013 ACM/IEEE International Symposium on, IEEE, October 2013, pp. 15-24.

[22] M. Linares-Vásquez, C. Vendome, Q. Luo and D. Poshyvanyk, “How developers detect and fix performance bottlenecks in Android apps”. In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on, IEEE, September 2015, pp. 352-361.

[23] R. Coelho, A. Rashid, A. Garcia, F. Ferrari, N. Cacho, U. Kulesza, A. von Staa and C. Lucena, “Assessing the impact of aspects on exception flows: An exploratory study. In European Conference on Object-Oriented Programming, 2008, pp. 207-234.