Sobre a Percepção dos Desenvolvedores Android em Relação aos Maus Cheiros de Código

Suelen G. Carvalho
Universidade de São Paulo
Rua do Matão, 1010
São Paulo, SP 05508-090
suelengc@ime.usp.br

Marco Aurélio Gerosa
Northern Arizona University
E Runke Dr
Flagstaff, Arizona 86011
gerosa@ime.usp.br

Maurício Aniche
Delft University of Technology
Mekelweg 2
Delft, The Netherlands 2628 CD
m.f.aniche@tudelft.nl

ABSTRACT

Cheiros de código são fortes aliados na busca pela qualidade de código durante o desenvolvimento de software pois possibilitam a implementação de ferramentas de detecção automática de trechos de códigos problemáticos ou mesmo a inspeção manual. Apesar de já existirem muitos cheiros de código catalogados, pesquisas sugerem que tecnologias diferentes podem apresentar cheiros de código específicos, e uma tecnologia que tem chamado a atenção de muitos pesquisadores é o Android. Neste artigo, nós investigamos a existência de maus cheiros em projetos Android. Nós conduzimos um *survey* com 45 desenvolvedores e descobrimos que além de maus cheiros já mapeados, algumas estruturas específicas da plataforma são amplamente percebidas como más práticas, portanto, possíveis cheiros de código específicos. Desta percepção propômos três cheiros de código Android, validados com um especialista e em um experimento com 30 desenvolvedores. Ao final, discutimos os resultados encontrados bem como pontos de melhoria e trabalhos futuros.

KEYWORDS

Android, cheiros de código, qualidade de código

ACM Reference format:
Suelen G. Carvalho, Marco Aurélio Gerosa, and Maurício Aniche. 2017. Sobre a Percepção dos Desenvolvedores Android em Relação aos Maus Cheiros de Código. In *Proceedings of SBES 2017: 31st Brazilian Symposium on Software Engineering, Fortaleza, Ceará Brasil, Setembro 2017 (SBES'17),* 10 pages.
DOI: 10.475/123_4

1 INTRODUÇÃO

Escrever código com qualidade tem se tornado cada vez mais importante com o aumento da complexidade da tecnologia. Existem diferentes técnicas que auxiliam os desenvolvedores a escrever código com qualidade incluindo *design patterns* e cheiros de código. A falta de qualidade pode resultar em defeitos de software que podem custar a empresas quantias significativas, especialmente quando conduzem a falhas de software [12, 31]. Evolução e manutenção de software também já se provaram como os maiores gastos com aplicações [39].

Uma das formas de aumentar a qualidade de software é identificar trechos de códigos ruins e refatorá-los, ou seja, alterar o código sem alterar o comportamento [18]. Com base nisso, temos que cheiros de código desempenham um importante papel na busca por qualidade de código, pois eles são sintomas que podem indicar problemas mais profundos no software, mas não necessariamente são o problema em si [19]. Seu mapeamento possibilita a definição de heurísticas que, por sua vez, possibilitam a implementação de ferramentas que os identificam de modo automático no código. PMD [8], Checkstyle e FindBugs são exemplos de ferramentas que identificam automaticamente alguns tipos de cheiros de código em códigos Java.

Determinar o que é ou não um cheiros de código é subjetivo e pode variar de acordo com tecnologia, desenvolvedor, metologia de desenvolvimento dentre outros aspectos [9]. Alguns estudos têm buscado por cheiros de código tradicionais em projetos Android. Por exemplo, Verloop [40] analisou se classes derivadas do SDK Android são mais ou menos propensas a cheiros de código tradicionais do que classes puramente Java. Linares et al. [27] usaram o método DECOR para realizar a detecção de 18 *anti-patterns* orientado a objetos em aplicativos móveis. Outros estudos identificaram cheiros de código específicos Android relacionados ao consumo inteligente de recursos do dispositivo, como bateria e memória, usabilidade, dentre outros [21, 32].

Nossa pesquisa complementa as anteriores no sentido de que também buscamos cheiros de código Android, e se difere delas pois estamos buscando cheiros de código relacionados à qualidade do código Android, ou seja, qualidade relacionada a códigos específicos dessa plataforma. Por exemplo ACTIVITYS, FRAGMENTS e ADAPTERS são classes usadas na construção de telas e LISTENERS são responsáveis pelas interações com os usuários. Buscamos entender, por exemplo, *"quais são as **boas e más** práticas ao lidar com* ACTIVITYS, FRAGMENTS, ADAPTERS *e* LISTENERS*?"* ou *"quais são as **boas e más** práticas para a construção da interface visual?"*.

Optamos por focar em elementos relacionados ao *front-end* Android pois encontramos pesquisas com uma curiosidade similar, porém relacionadas a identificação de cheiros de código em tecnologias usadas no *front-end* de projetos web [11, 17, 20]. E enquanto que cheiros de código em projetos

SBES'17, Fortaleza, Ceará Brasil

DOI: 10.475/123_4

Java já foram extensivamente estudados [18, 24, 30], o *front-end* Android ainda carece de estudo e possui peculiaridades não encontradas em código Java tradicional [28]. Alguns exemplos dessas peculiaridas são o ciclo de vida de ACTIVITYS e FRAGMENTS e a criação da interface visual que é feita através de arquivos XML chamados de LAYOUT RESOURCES.

Para definirmos quais elementos representam o *front-end* Android fizemos uma extensa revisão da documentação oficial [37] e chegamos nos seguintes itens: ACTIVITYS, FRAGMENTS, LISTENERS, ADAPTERS e os recursos do aplicativo, que são arquivos XML ou imagens utilizados na interface visual como por exemplo DRAWABLES, LAYOUTS, STYLES e COLORS. Como existem muitos tipos de recursos do aplicativo [38], com o objetivo de limitar o tamanho do questionário e foco da pesquisa, selecionamos quatro: LAYOUT, STYLES, STRING e DRAWABLE. Optamos por esses recursos pois os mesmos estão presentes no template padrão do Android Studio [36], IDE oficial para desenvolvimento de projetos da plataforma Android [2].

Para obter os dados iniciais, publicamos um questionário online com perguntas sobre boas e más práticas no desenvolvimento do *front-end* Android. Para possibilitar respostas mais completas foram usadas perguntas dissertativas. Com isso, pretendemos responder as seguintes questões de pesquisa:

RQ1 O que desenvolvedores consideram boas e más práticas no desenvolvimento Android?

RQ2 Códigos afetados por estas más práticas são percebidos pelos desenvolvedores como problemáticos?

As seções seguintes deste artigo estão organizadas da seguinte forma: na Seção 2 discutimos trabalhos relacionados e o estado da arte sobre Android e cheiros de código. Na Seção 3 falamos sobre a metodologia de pesquisa utilizada. A Seção 4 apresenta os resultados obtidos. Na Seção 5 discutimos os resultados. Na Seção 6 tratamos das ameaças à validade do nosso estudo e então, na Seção 7 concluímos.

2 TRABALHOS RELACIONADOS

Muitas pesquisas têm sido realizadas sobre a plataforma Android, muitas delas focam em vulnerabilidades [13–15, 26, 42, 45, 46], autenticação [16, 43, 44] e testes [10, 23]. Diferentemente dessas pesquisas, nossa pesquisa tem foco na percepção dos desenvolvedores sobre boas e más práicas de desenvolvimento na plataforma Android.

A percepção desempenha um importante papel na definição de code smells relacionados a uma tecnologia, visto que code smells possuem uma natureza subjetiva. Code smells desempenham um importante papel na busca por qualidade de código, visto que, após mapeados, podemos chegar a heurísticas para identificá-los e com essas heurísticas, implementar ferramentas que automatizem o processo de identificar códigos maus cheirosos.

Verloop [40] conduziu um estudo no qual avaliou por meio de 4 ferramentas de detecção automatizada de cheiros de código (JDeodorant, Checkstyle, PMD e UCDetector) a presença de 5 cheiros de código (Long Method, Large Class, Long Parameter List, Feature Envy e Dead Code) em 4 projetos Android. Nossa pesquisa se relaciona com a de Verloop no sentido de que também estamos buscando por cheiros de código, entretanto, em vez de de buscarmos por cheiros de código já definidos, realizamos uma abordagem inversa na qual, primeiro buscamos entender a percepção de desenvolvedores sobre boas e más práticas em Android, e a partir dessa percepção, relacionamos com algum cheiro de código pré-existente ou derivamos algum novo.

Gottschalk et al [21] conduziram um estudo sobre formas de detectar e refatorar cheiros de código relacionados ao uso eficiênte de energia. Os autores compilaram um catálogo com 8 cheiros de código e trabalharam sob um trecho de código Android para exemplificar um deles, o "binding resource too early", quando algum recurso é alocado muito antes de precisar ser utilizado. Essa pesquisa é relacionada à nossa por ambas considerarem a tecnologia Android e se diferenciam pois focamos na busca por cheiros de código relacionados a qualidade de código, no sentido de legibilidade e manutenablidade.

Aplicativos Android são escritos na linguagem de programação Java [33]. Então a primeira questão é: por que buscar por *smells* Android sendo que já existem tantos *smells* Java? Pesquisas têm demonstrado que tecnologias diferentes podem apresentar *code smells* específicos, como por exemplo Aniche et al. identificou 6 *code smells* específicos ao framework Spring MVC, um framework Java para desenvolvimento web. Outras pesquisas concluem que projetos Android possuem características diferentes de projetos java [22, 28, 32], por exemplo, o *front-end* é representado por arquivos XML e o ponto de entrada da aplicação é dado por *event-handler* [1] como o método ONCREATE. Encontramos também diversas pesquisas sobre *code smells* sobre tecnologias usadas no desenvolvimento de *front-end* web como CSS [20] e JavaScript [17]. Essas pesquisas nos inspiraram a buscar entender se existem *code smells* no *front-end* Android.

[seção não finalizada, á concluir.]

3 METODOLOGIA

Conduzimos um estudo qualitativo e exploratório onde os dados foram coletados através de um questionário online com desenvoveldores Android. Esta seção descreve de forma detalhada a estrutura do questionário, os participantes e a análise realizada sobre as respostas obtidas.

3.1 Questionário

O questionário continha 25 questões divididas em três seções. A primeira seção continha 6 perguntas demográficas, a segunda seção continha 16 perguntas sobre boas e más práticas relacionadas ao *front-end* Android e a terceira seção continha 3 perguntas, 2 para obter últimos pensamentos sobre boas e más práticas e 1 solicitando email caso o participante tivesse

interesse em etapas futuras da pesquisa. O questionário foi escrito em inglês porém informava o participante que respostas em inglês ou português eram aceitas. Antes da divulgação, realizamos um piloto com 3 desenvolvedores Android. Todos estes dados estão disponíveis no nosso pacote de replicação[1].

A primeira seção continha 6 questões demográficas obrigatórias de multipla escolha. Abordavam sobre idade (18 ou menos, 19 a 24, 25 a 34 e assim por diante até 55 ou mais), estado de residência (foi dada uma lista com estados do Brasil, Estados Unidos e Europa), anos de experiência com desenvolvimento de software, (1 anos ou menos, 2 anos, 3 anos, e assim por diante até 10 ou mais), anos de experiência com desenvolvimento Android (mesma escala de anos da questão anterior), uma questão sobre linguagens que o participante se considerava proeficiente (Java, Python, Ruby, Android, dentre outras) e sobre o último grau de escolaridade (estudante de bacharelado, bacharelado, mestrado e doutorado). As questões sobre idade, região, linguagens e grau de escolaridade continham a opção "outros" para o caso de nenhuma das opções atenderem, ao selecionar "outros" era possível escrever uma resposta.

A segunda seção continha 16 questões opcionais e dissertativas sobre boas e más práticas relacionadas ao *front-end* Android. Para cada elemento do *front-end* Android foram feitas duas perguntas, uma sobre boas e outra sobre más práticas percebidas pelos participantes. Por exemplo, para o elemento ACTIVITY foram feitas as seguintes perguntas:

* Você tem alguma boa prática para lidar com Activities?
* Você considera alguma coisa uma má prática ao lidar com Activities?

A terceira seção continha 3 perguntas opcionais e dissertativas, 2 para captar qualquer última ideia sobre boas e más práticas não captadas nas questões anteriores e 1 solicitando o email do participante caso o mesmo tivesse interesse em participar de etapas futuras da pesquisa.

Antes da divulgação, realizamos um piloto com 3 desenvolvedores Android e com o feedback deles fizemos alguns ajustes relacionados a obrigatoriedade das perguntas da segunda seção do questionário, onde todas tornaram-se opcionais. As respostas dos participantes piloto foram desconsideradas para efeitos de viés.

Todas as 18 questões sobre boas e más práticas (16 na segunda seção e 2 da terceira) são apresentadas na Tabela 1.

3.2 Participantes

O questionário foi divulgado em redes sociais como Facebook, Twitter e Linkedin, em grupos de discussão sobre Android como *Android Dev Brasil*, *Android Brasil Projetos* e o grupo do *Slack Android Dev Br*, maior grupo de desenvolvedores Android do Brasil com 2622 participantes até o momento da escrita deste artigo.

O questionário esteve aberto por aproximadamente 3 meses e meio, de 9 de Outubro de 2016 até 18 de Janeiro de 2017. Recebemos um total de 45 respostas sendo que 41 foram submetidas em Outubro, 3 no começo de Novembro e 1 em Janeiro. Uma possível explicação para praticamente não termos tido respostas nos meses de Novembro e Dezembro pode ser as festas comemorativas de final de ano.

80% dos participantes responderam pelo menos 3 perguntas sobre boas e más práticas no *front-end* Android (7 responderam de 3 a 6, 6 responderam de 8 a 10 e 23 responderam 13 ou mais, sendo que desses, 14 responderam todas) e apenas 20% responderam uma (2 participantes) ou nenhuma (7 participantes). A pergunta solicitando o email foi respondida por 53% dos participantes, o que pode indicar um interesse legítimo da comunidade de desenvolvedores Android pelo tema, reforçando a relevância do estudo. A Tabela 1 apresenta a quantidade de respostas obtidas por cada uma das 18 perguntas sobre boas e más práticas, podemos notar que a pergunta mais respondida foi a Q1 e a menos respondida foi a Q18.

Com a análise das questões demográficas foi possível notar que atingimos com sucesso *desenvolvedores Android com variados níveis de experiência e de diversas regiões* pois: 1) 100% dos participantes indicaram possuir alguma experiência com desenvolvimento Android, 2) menos de 14% indicaram possuir 1 anos ou menos de experiência com Android e mais de 86% indicaram 2 anos ou mais (15,5% 2 anos, 13,3% 4 anos, 6,5% 5 anos, 15,5% 6 anos, 4,4% 7 anos e 4,4% 8 anos), 4) 36 respostas foram do Brasil, 7 de países europeus e 1 dos Estados Unidos (Califórnia). Vale lembrar que a plarforma Android completa 10 anos em 2017, ou seja, 5 anos de experiência nessa plataforma representa 50% do tempo de vida dela desde seu anúncio em 2007. Os dados sobre a experiência dos participantes são apresentados na Tabela 2.

Anos de Experiência	**Participantes**	**%**
1 ano ou menos	6	13,3 %
2 anos	7	15,6 %
3 anos	12	26,7 %
4 anos	6	13,3 %
5 anos	3	6,7 %
6 anos	7	15,6 %
7 anos	2	4,4 %
8 anos	2	4,4 %

Tabela 2: Experiência dos participantes com desenvolvimento Android.

3.3 Análise dos Dados

O processo de análise partiu da listagem das 45 respostas do questionário e se deu em 4 passos: verticalização, limpeza dos dados, codificação e divisão.

O processo que denominamos de *verticalização* consistiu em considerar cada resposta de boa ou má prática como um registro individual a ser analisado. Ou seja, cada participante

[1] https://github.com/SuelenGC/android-code-smells-article

Id	Questões	Respostas		Participantes
		Total	%	
Q1	Você tem alguma boa prática para lidar com Activities?	36	80%	P1, P2, P4–P12, P14–P17, P19, P22, P23, P25–P32, P34–P37, P39–P43, P45
Q2	Você considera alguma coisa uma má prática ao lidar com Activities?	35	78%	P2, P4–P11, P14–P17, P19, P22, P23, P25–P32, P34–P37, P39–P45
Q3	Você tem alguma boa prática para lidar com Fragments?	33	73%	P4–P11, P14–P17, P19, P22, P23, P25–P28, P30–P32, P34–P37, P39–P45
Q4	Você considera alguma coisa uma má prática ao lidar com Fragments?	31	69%	P2, P4–P11, P14, P15, P17, P19, P22, P23, P25–P28, P31,P32, P34–P37, P39–P43, P45
Q5	Você tem alguma boa prática para lidar com Adapters?	30	67%	P2, P4–P11, P14, P15, P17–P19, P22, P23, P26, P28, P29, P31,P32, P34–P37, P39–P43, P45
Q6	Você considera alguma coisa uma má prática ao lidar com Adapters?	27	60%	P2, P4–P8, P10, P11, P14, P18, P19, P22, P23, P26, P28, P31, P34–P37, P39–P45
Q7	Você tem alguma boa prática para lidar com Listeners?	24	53%	P2, P4–P6, P8, P9, P11, P14, P22, P23, P26, P28, P29, P31, P32, P34, P36, P37, P39–P43, P45
Q8	Você considera alguma coisa uma má prática ao lidar com Listeners?	23	51%	P2, P4, P5, P8, P9, P11, P14, P19, P22, P23, P26, P28, P31, P32, P34, P36, P37, P39–P44
Q9	Você tem alguma boa prática para lidar com Layout Resources?	28	62%	P4–P9, P11, P14, P19, P22, P23, P26–P29, P31, P32, P34–P37, P39–P45
Q10	Você considera alguma coisa uma má prática ao lidar com Layout Resources?	23	51%	P4, P5, P7–P9, P11, P22, P23, P26, P28, P31, P32, P34–P37, P39–P45
Q11	Você tem alguma boa prática para lidar com Styles Resources?	23	51%	P4–P9, P11, P18, P22, P23, P26, P28, P31, P32, P34–P37, P39–P43
Q12	Você considera alguma coisa uma má prática ao lidar com Styles Resources?	22	49%	P4–P8, P11, P18, P22, P23, P26, P28, P31, P32, P34–P37, P39–P43
Q13	Você tem alguma boa prática para lidar com String Resources?	28	62%	P4–P6, P8–P11, P14, P18, P22, P23, P26–P29, P31, P32, P34–P37, P39–P45
Q14	Você considera alguma coisa uma má prática ao lidar com String Resources?	23	51%	P4–P6, P8, P9, P11, P14, P18, P22, P23, P26, P28, P31, P32, P34–P37, P40–P43, P45
Q15	Você tem alguma boa prática para lidar com Drawable Resources?	24	53%	P4–P6, P8–P11, P14, P18, P22, P23, P26, P28, P31, P32, P34–P37, P39–P43
Q16	Você considera alguma coisa uma má prática ao lidar com Drawable Resources?	21	47%	P4–P6, P8, P11, P14, P18, P22, P23, P26, P28, P31, P32, P34, P36, P37, P40–P44
Q17	Existem outras *BOAS* práticas sobre a Camada de Apresentação Android que nós não perguntamos ou que você não disse ainda?	22	49%	P2, P4, P8, P10, P11, P14, P18, P22, P23, P26, P28, P31, P32, P34, P36, P37, P39–P43, P45
Q18	Existem outras *MÁS* práticas sobre a Camada de Apresentação Android que nós não perguntamos ou que você não disse ainda?	20	44%	P2, P4, P8, P10, P11, P18, P22, P23, P28, P31, P32, P34, P36, P37, P40–P45

** Os participantes P3, P13, P20, P21, P24, P33 e P38 não responderam nenhuma das questões da segunda e terceira seção.*

Tabela 1: Total de respostas obtidas por cada questão sobre boas e más práticas no *front-end* Android.

respondeu 18 perguntas sobre boas e más práticas no *front-end* Android (2 perguntas para cada elemento e mais duas perguntas genéricas). Com o processo de *verticalização*, cada uma dessas respostas se tornou um registro, ou seja, cada participante resultava em 18 respostas a serem analisadas, totalizando 810 respostas (18 perguntas multiplicado por 45 participantes) sobre boas e más práticas.

O passo seguinte foi realizar a *limpeza dos dados*. Esse passo consistiu em remover respostas obviamente não úteis como respostas em branco, que continham frases como *"Não"*, *"Não que eu saiba"*, *"Eu não me lembro"* e similares, as consideradas vagas como *"Eu não tenho certeza se são boas praticas mas uso o que vejo por ai"*, as consideradas genéricas como *"Como todo código java..."* e as que não eram relacionadas a boas práticas de código. Das 810 boas e más práticas, 352 foram consideradas e 458 desconsideradas. Das 352, 44,6% foram apontadas como más práticas e 55,4% como boas práticas.

Em seguida, realizamos a codificação sobre as boas e más práticas. O processo de codificação consistiu em analisar cada resposta e atribuir uma ou mais categorias. Durante esse processo, houveram 30 respostas que não eram triviais de identificar uma categoria ou mesmo de dizer se essas respostas deveriam ser consideradas. Essas respostas foram marcadas como *"talvez"* e reavaliadas ao final, onde 6 permaneceram e 24 foram desconsideradas. Ainda durante a codificação, 9 respostas incialmente consideradas, foram desconsideradas. Para toda resposta desconsiderada nesse passo, foi indicado um motivo. Ao final da codificação, as categorias foram agrupadas por temas.

Por último realizamos o passo de divisão. Esse passo consistiu em dividir as respostas que receberam mais de uma categoria em duas ou mais respostas, de acordo com o número de categorias identificadas, de forma a resultar em uma categoria por resposta. Por exemplo, a resposta *"Não fazer Activities serem callbacks de execuções assíncronas. Herdar sempre das classes fornecidas pelas bibliotecas de suporte,*

nunca diretamente da plataforma" indica na primeira oração uma categoria e na segunda oração, outra categoria. Ao dividí-la, mantivemos apenas o trecho da resposta relativo a categoria, como se fossem duas respostas distintas e válidas. Em algumas divisões realizadas, a resposta completa era necessária para entender ambas as categorizações, nesses casos, mantivemos a resposta original, mesmo que duplicada, e categorizamos cada uma de forma diferente.

Ao final da análise constavam 389 respostas individualmente categorizadas sobre boas e más práticas no *front-end* Android.

4 RESULTADOS

Durante o processo de codificação das 18 perguntas sobre boas e más práticas, 56 categorias emergiram. Classificamos as categorias de acordo com sua recorrência, ou seja, a quantidade de respostas a qual ela foi atribuída. Utilizamos a seguinte escala: baixíssima recorrência significa menos de 3 respostas, baixa recorrência significa de 3 a 7 respostas, média recorrência significa de 8 a 20 respostas e alta recorrência significa acima de 20 respostas.

Na Figura 1 é possível observar que o maior conjunto de categorias é o de baixíssima recorrência, ou seja, pouquíssimos participantes comentaram sobre esses assuntos. Em seguida, temos o conjunto de média, baixa e alta recorrência, respectivamente.

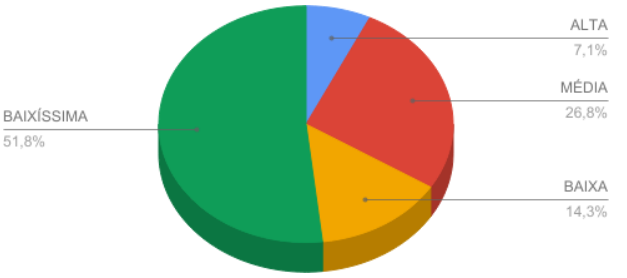

Figura 1: Distribuição das Categorias x Recorrência

A Tabela 3 apresenta o número de ocorrências das categorias de alta, média e baixa recorrência em cada questão relacionada a boas e más práticas. A última linha da tabela, #Categorias, apresenta quantas categorias emergiram de cada questão, como cada questão está diretamente ligada a um elemento do *front-end* Android, podemos interpretá-la da seguinte forma: **quais são os pontos de atenção a serem analisados em determinado elemento Android?** A última coluna da tabela, #Q, apresenta em quantas questões cada categoria surgiu, podemos interpretá-la da seguinte forma: **com base na categoria, quais elementos devem ser investigados**?

Esta seção está organizada em 4 subseções, respectivamente abordando sobre as categorias de alta, média, baixa e baixíssima recorrência. A quantidade de respostas recebida por cada categoria é apresentada em detalhes na Tabela 3.

4.1 Categorias de Alta Recorrência

Obtivemos 4 categorias consideradas de alta recorrência: Lógica em Views, Padrão de Nome de Recursos, Recursos Mágicos e Views Aninhados.

4.1.1 Lógica em Views. Esta categoria reúne respostas que indicam como má prática haver regras de negócio nos elementos Activitys, Fragments, Listeners e Adapters. De forma similar, respostas indicam como boas práticas que esses elementos apenas contenham códigos relacionados a interface com o usuário, e para isso sugerem o uso de alguns padrões, foram eles: MVP (*Model-View-Presenter*) [4, 5], MVVM (*Model-View-ViewModel*) [6] e *Clean Architecture* [29]. Exemplos de frases que indicaram más práticas são: P16 sobre Activitys diz *"Fazer lógica de negócio"* (tradução livre), P19 diz *"Colocar regra de negócio no adapter"* e P11 diz *"Manter lógica de negócio em Fragments"* (tradução livre). Exemplos de frases que indicaram boas prática são: P16 diz sobre Activitys *"Elas representam uma única tela e apenas interagem com a UI, qualquer lógica deve ser delegada para outra classe"* (tradução livre), P23 diz *"Apenas código relacionado à Interface de Usuário nas Activities"*, P40 diz *"Adapters devem apenas se preocupar sobre como mostrar os dados, sem trabalhá-los"*, P2 diz *"As activitys que eu crio normalmente tem um propósito único e estado básico [...] eu uso MVP a maior parte do tempo, então minhas activitys normalmente representam uma view no MVP"*. Os elementos afetados por essa categoria são: Activitys, Fragments, Listeners e Adapters.

4.1.2 Padrão de Nome de Recursos. Esta categoria reúne respostas que indicam como má prática o não uso de um padrão de nomenclatura a ser usado nos recursos da aplicação. De forma similar, respostas indicam como boas práticas o uso de um padrão de nomenclatura a ser usados nos recursos. Exemplos de frases que indicaram más práticas são: P8 sobre Style Resources diz *"[...] o nome das strings sem um contexto"* (tradução livre), P37 também sobre Style Resources diz *"Nada além de ter uma boa convenção de nomes"* (tradução livre), ainda P37, porém sobre Layout Resources diz *"Mantenha uma convenção de nomes da sua escolha [...]"* (tradução livre). Exemplos de frases que indicaram boas prática são: P27 diz sobre String Resources *"Iniciar o nome de uma string com o nome da tela onde vai ser usada"*, P43 sobre Layout Resources diz *"Ter uma boa convenção de nomeação"* (tradução livre), P11 diz sobre Style Resources *"[...] colocar um bom nome [...]"* (tradução livre). Os elementos que entraram nessa categoria foram: Activitys, Layout Resources, String Resources, Style Resources e Drawable Resources.

Dentre as respostas, algumas indicaram padrões de preferência. P11 indica usar prefixos nos Layout Resources: `activity_`, `fragment_`, `ui_` (para UI customizadas). P12 sugeriu usar sufixos em Activitys: `_Activity`. Os padrões indicados para String Resources foram: P27 indicou *"Iniciar o nome da string com o nome da tela onde vai ser usada"*, P6 sugeriu a convenção `[screen]_[type]_[text]` e citou como

Categoria	#T	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	#Q
ALTA RECORRÊNCIA																				
Lógica em Views	53	12	15	6	8	3	8	–	1	–	–	–	–	–	–	–	–	–	–	7
Padrão de Nome de Recursos	24	1	–	–	–	–	–	–	–	3	2	3	2	8	2	3	–	–	–	8
Recursos Mágicos	23	–	–	–	–	–	–	–	–	4	2	1	1	9	6	–	–	–	–	6
Views Aninhados	21	–	–	–	–	1	–	–	–	9	9	–	–	–	–	–	–	1	1	5
MÉDIA RECORRÊNCIA																				
Acoplamento Entre Views	18	–	2	4	6	–	3	1	2	–	–	–	–	–	–	–	–	–	–	6
Ciclo de Vida	16	4	3	3	5	–	–	1	–	–	–	–	–	–	–	–	–	–	–	5
Use Include	15	–	–	–	–	–	–	–	–	12	2	–	–	–	–	–	–	1	–	3
Padrão View Holder	14	–	–	–	–	12	2	–	–	–	–	–	–	–	–	–	–	–	–	2
Tamanho de Imagens Importam	12	–	–	–	–	–	–	–	–	1	1	–	–	–	–	4	6	–	–	4
Comportamento Suspeito	14	1	1	–	–	1	3	5	3	–	–	–	–	–	–	–	–	–	–	6
Classe Deus/Longa*	11	1	4	1	2	–	1	–	1	–	–	–	1	–	–	–	–	–	–	7
Use Fragment	11	3	2	5	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	4
Use Imagens Vetoriais	11	–	–	–	–	–	–	–	–	–	–	–	–	–	–	11	–	–	–	1
Fragment Apenas Se Necessário	10	–	–	8	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2
Use Arquiteturas Conhecidas	9	–	–	1	–	2	–	–	–	–	–	–	–	–	–	–	–	5	1	4
Recurso de Estilo Deus	8	–	–	–	–	–	–	–	–	–	–	5	3	–	–	–	–	–	–	2
Recurso de Strings Bagunçado	8	–	–	–	–	–	–	–	–	–	–	–	–	4	4	–	–	–	–	2
Atributos de Estilos Repetidos	8	–	–	–	–	–	–	–	–	1	2	2	2	–	–	–	–	1	–	5
BAIXA RECORRÊNCIA																				
Activity Inexistente	7	2	4	–	–	–	–	–	1	–	–	–	–	–	–	–	–	–	–	3
Evite Imagens	7	–	–	–	–	–	–	–	–	1	–	–	–	–	–	4	2	–	–	3
Reuso Excessivo de Strings	6	–	–	–	–	–	–	–	–	–	–	–	–	2	4	–	–	–	–	2
Adapter Flexível	6	–	–	–	–	3	2	–	–	–	1	–	–	–	–	–	–	–	–	3
Herança**	5	2	–	2	–	1	–	–	–	–	–	–	–	–	–	–	–	–	–	3
Listener Escondido	3	–	–	–	–	–	–	–	3	–	–	–	–	–	–	–	–	–	–	1
Operações de IO	6	–	3	–	2	–	1	–	–	–	–	–	–	–	–	–	–	–	–	3
#Totais	35	37	35	26	23	20	11	11	31	19	11	9	23	16	22	8	11	2		
#Categorias	10	10	9	7	7	7	4	6	7	7	4	5	4	4	4	2	6	2		

* Classe Deus [24] e Classe Longa [7] são cheiros de código tradicionais previamente definidos em literaturas.

** Herança é um conceito da Programação Orientada a Objetos [41].

Coluna **#T**: recorrência geral da categoria.

Coluna **#Q**: total de questões distintas em cada categoria.

Linha **#Totais**: total de respostas obtidas em cada questão.

Linha **#Categorias**: total de categorias distinstas de cada questão.

Tabela 3: Lista de categorias de alta, média e baixa recorrência.

exemplo `welcome_message_title`. P34 indicou que deve-se usar como prefixo o recurso usando a string, por exemplo `dialog.STRING_NAME` ou `hint.STRING_NAME`. De forma similar porém sem sugerir um exemplo, P4 sugeriu basear o nome da string no nome do recurso que a esta usando. Não foram sugeridos nenhum padrão para Styles Resources e Drawable Resources.

4.1.3 Recursos Mágicos. Esta categoria reúne respostas que indicam como má prática o uso direto de valores como, por exemplo, strings, números e cores, sem a criação um recurso. De forma similar, respostas indicam como boas práticas o uso de um padrão de nomenclatura a ser usados nos recursos. O nome dessa categoria foi inspirado no cheiro de código *Magic Number* [30] que trata sobre números usados diretamente no código. Exemplos de frases que indicaram más práticas são: P23 diz *"Strings diretamente no código"*, P31 e P35 falam respectivamente sobre não extrair as strings e sobre não extrair os valores dos arquivos de layout. Exemplos de frases que indicaram boas prática são: P7 diz *"Sempre pegar valores de string ou dp de seus respectivos resources para facilitar"*, P36 diz para *"sempre adicionar as strings em resources para traduzir em diversos idiomas [...]"*. Os elementos que entraram nessa categoria foram: Layout Resources, String Resources e Style Resources.

4.1.4 Views Aninhados. Esta categoria reúne respostas que indicam como má prática o uso de profundos aninhamentos na construção de layouts. De forma similar, respostas indicam como boas práticas evitar ao máximo o aninhamento de *views*. Exemplos de frases que indicaram más práticas são: P26 diz *"Hierarquia de views longas"* (tradução livre), P4 aborda a

mesma ideia ao dizer *"Estruturas profundamente aninhadas"* (tradução livre), P39 diz *"Hierarquias desnecessárias"* e P45 diz *"Criar muitos ViewGroups dentro de ViewGroups"*. Exemplos de frases que indicaram boas prática são: P4 diz *"tento usar o mínimo de layout aninhado"*, P19 diz *"Utilizar o mínimo de camadas possível"*, P8 diz *"[...] não fazer uma hierarquia profunda de ViewGroups [...]"*. Apenas o elemento Layout Resources recebeu esta categoria. O site oficial do Android conta com informações e ferramentas automatizadas para lidar com esse sintoma [35].

4.2 Categorias de Média Recorrência

Obtivemos 16 categorias consideradas de média recorrência: Acoplamento Entre Views, Padrão MVP, Ciclo de Vida, Use Include, Padrão View Holder, Comportamento Suspeito, Tamanho de Imagens Importam, Classe Deus/Longa, Use Fragment, Use Imagens Vetoriais, Fragment Apenas Se Necessário, Use Arquiteturas Conhecidas, Recurso de Estilo Deus, Recurso de Strings Bagunçado e Atributos de Estilos Repetidos.

4.2.1 Acoplamento Entre Views. Esta categoria reúne respostas que indicam como má prática o acoplamento entre Activitys, Fragments, Adapters e Listeners, ou seja, a existência de referências diretas entre elas. De forma similar, respostas indicam como boas práticas que estas classes não se conheçam diretamente. Com base nas respostas, identificamos 3 situações onde essa má prática é percebida.

O primeira situação é quando o Fragment está acoplado à Activitys, outros Fragments ou componentes. Sobre o acoplamento de Fragments com Activitys, P19 diz *"Acoplar o fragment a activity ao invés de utilizar interfaces é uma prática ruim"*. P10, P31 e P45 indicam como má prática *"acoplar o* Fragment *com a* Activity*"* (tradução livre). Sobre o acoplamento de Fragments com outros Fragments, P37 diz que *"Fragments nunca devem tentar falar uns com os outros diretamente"* e P45 diz *"[é uma má prática] integragir com outro Fragment diretamente"* (tradução livre). Sobre o Fragments serem acoplados a outros componentes, P6 diz *"Seja um componente de UI reutilizável. Então evite dependência de outros componentes da aplicação"*. Como boa prática, para a comunicação entre essas classes, são indicados: o uso de *interfaces*, o método onAttach existente em Fragments (este método é disparado pelo Android ao associar um Fragment a uma Activity) ou a biblioteca *EventBus* [3]. P36 diz *"Criar uma interface para a comunicação entre Activity e Fragment, ou utilizar o EventBus."* e P44 diz *"Use e abuse do método onAttach para se comunicar com Activity"*.

A segunda situação é quando o Listener está acoplado à Activitys. P40 diz que é uma má prática *"[o* Listener*] conter uma referência forte à Activitys"* (tradução livre), P4 exprime a mesma ideia com uma frase um pouco diferente.

A terceira situação é quando o Adapter está acoplado à Activitys ou Fragments. P10 indicou como má prática em *Adapters* o *"alto acoplamento com a Activity"* e P45 exprime a mesma ideia ao dizer *"Acessar Activitys ou Fragments diretamente"*.

Os elementos que entraram nessa categoria foram: Activitys, Fragments, Listeners e Adapters.

4.2.2 Ciclo de Vida. Esta categoria reúne respostas que indicam como má prática o uso incorreto do ciclo de vida Activitys e Fragments. De forma similar, respostas indicam como boas práticas respeitar o ciclo de vida desses elementos e não confundir o ciclo de vida de ambos. Exemplos de frases que indicaram más práticas são: P23 diz *"código que depende de estado e não se adapta ao ciclo de vida das Activities."*, P28 diz *"Erros ao interpretar o ciclo de vida"*, P8 diz *"considerar o ciclo de vida de fragments como os de activitys"* (tradução livre). Exemplos de frases que indicaram boas prática são: P43 diz *"Conhecer seu [da activity] ciclo de vida"*, P28, P31 e P15 falam *"tomar cuidado e respeitar o ciclo de vida de* fragments *e* activitys*"*. Os elementos que entraram nessa categoria foram: Activitys e Fragments.

4.2.3 Use Include. Esta categoria reúne respostas que indicam como má prática arquivos Layout Resources grandes e complexos. Respostas indicam como boas práticas a quebra desses arquivos em vários outros e o uso da *tag* de layout `include` para uni-los. Exemplos de frases que indicaram más práticas são: P41 diz *"Copiar e colar trechos similares de tela, sem usar includes"* (tradução livre), P23 diz *"[...] utilizar muitos recursos no mesmo arquivo de layout"*. Exemplos de frases que indicaram boas prática são: P34 diz *"eu apenas tento reusá-los através do uso de includes"*, P40 diz *"modularize-os"* (tradução livre), P9 diz *use includes para simplificar multiplas configurações [de tela]* (tradução livre).

4.2.4 Padrão View Holder. Esta categoria reúne respostas que indicam como má prática o não uso do padrão *View Holder* [34] para melhorar o desempenho de listagens. De forma similar, respostas indicam como boas práticas evitar o uso do padrão *View Holder*. Exemplo de frase que indicou má prática é P8 ao dizer *"[...] evitar o padrão ViewHolder"* (tradução livre). Exemplos de frases que indicaram boas prática são: P36 diz *"Reutilizar a view utilizando ViewHolder."*, de forma similar P39 diz *"Usar o padrão ViewHolder"*. P45 sugere o uso do RecyclerView, um elemento Android para a construção de listas que já implementa o padrão *ViewHolder* [34]. Apenas o elemento Adapter entrou nessa categoria.

4.2.5 Tamanho de Imagens Importam. Esta categoria reúne respostas que indicam como má prática ter apenas uma imagem para atender a todas as resoluções. De forma similar, respostas indicam como boas práticas ter a mesma imagem em diversos tamanhos para atender a resoluções diferentes. Exemplos de frases que indicaram más práticas são: P31 diz *"ter apenas uma imagem para multiplas densidades"* (tradução livre), P4 diz *"Baixar uma imagem muito grande quando não é necessário. Há melhores formas de usar memória"* (tradução livre), P44 diz *"Não criar [versões da] imagem para todos as resoluções"* (tradução livre). Exemplos de frases que indicaram boas prática são: P34 diz *"Nada especial, apenas mantê-las em seus respectivos diretórios e ter variados tamanhos delas"* (tradução livre), P36 diz *"Criar as pastas*

para diversas resoluções e colocar as imagens corretas". O único elemento que entrou nessa categoria é o que representa imagens, Drawable Resource.

4.2.6 Use Imagens Vetoriais. Esta categoria reúne 10 respostas (10 participantes) que indicam como boa prática o uso de Drawable Resources vetoriais. é recomendado sempre que possível usar imagens vetorias sobre outros tipos. Exemplos de frases que indicaram boas prática são: P28 diz *"Utilizar o máximo de Vector Drawables que for possível"*, P40 diz *"evite muitas imagens, use imagens vetoriais sempre que possível"*. O único elemento que entrou nessa categoria é o que representa imagens, Drawable Resource.

4.2.7 Operações de IO. Esta categoria reúne respostas que indicam como má prática realizar operações de IO, como consulta a banco de dados ou acesso a internet, a partir das classes Activitys, Fragments e Adapters. Respostas indicam como boas práticas que esses elementos lidem apenas com a interface com o usuário e sugerem para isso, o padrão MVP [4, 5]. Exemplos de frases que indicaram más práticas são: P26 sobre Activitys e Fragments diz *"fazer requests e consultas a banco de dados"* e sobre Adapter diz *"Fazer operações longas e requests de internet"* (tradução livre). P37 sobre Activitys diz *"Elas nunca devem fazer acesso a dados [...]"*. Exemplos de frases que indicaram boas prática são: P26 diz *"Fazer activities e fragments apenas lidar com ações da view, faça isso usando o [padrão] MVP"*. Os elementos que entraram nessa categoria foram: Activitys, Fragments e Adapters.

4.2.8 Recurso de Estilo Deus. Esta categoria reúne respostas que indicam como má prática o uso de apenas um arquivo para todos os Styles Resources. De forma similar, respostas indicam como boas práticas separar os estilos em mais de um arquio. Exemplos de frases que indicaram más práticas são: P28 diz *"Deixar tudo no mesmo arquivo styles.xml"*, P8 diz *"Arquivos de estilos grandes"* (tradução livre). Exemplos de frases que indicaram boas prática são: P28 diz *"Se possível, separar mais além do arquivo styles.xml padrão, já que é possível declarar múltiplos arquivos XML de estilo para a mesma configuração"*. P40 diz *"Divida-os. Temas e estilos é uma escolha racional"*. O único elemento que entrou nessa categoria foi o Style Resource.

4.2.9 Recurso de Strings Bagunçado. Esta categoria reúne respostas que indicam como má prática arquivos String Resources desorganizados ou o uso de apenas um arquivo para todos os String Resources. De forma similar, respostas indicam como boas práticas separar as *strings* em mais de um arquivo. Exemplos de frases que indicaram más práticas são: P28 diz *"Usar o mesmo arquivo strings.xml para tudo"*, P42 diz *"Não orgaizar as strings quando o strings.xml começa a ficar grande"*. Exemplos de frases que indicaram boas prática são: P28 diz *"Separar strings por tela em arquivos XML separados. Extremamente útil para identificar quais strings pertencentes a quais telas em projetos grandes"*. P32 diz *"Sempre busco separar em blocos, cada bloco representa uma activity e nunca aproveito uma String pra outra tela"*. O único elemento que entrou nessa categoria foi o String Resource.

4.2.10 Atributos de Estilos Repetidos. Esta categoria reúne respostas que indicam como má prática a repetição de atributos de estilo nos Layout Resource. De forma similar, respostas indicam como boas práticas sempre que identificar atributos repetidos, extraí-los para um estilo. Exemplos de frases que indicaram más práticas são: P32 diz *"Utilizar muitas propriedades em um único componente. Se tiver que usar muitas, prefiro colocar no arquivo de styles."*. Exemplos de frases que indicaram boas prática são: P34 diz *"Sempre que eu noto que tenho mais de um recurso usando o mesmo estilo, eu tento movê-lo para o meu style resource."* (tradução livre). Os elementos nessa categoria foram: Layout Resources e Style Resources.

4.3 Categorias de Baixa Recorrência

Obtivemos 8 categorias consideradas de baixa recorrência: Evite Imagens, Activity Inexistente, Ferramentas de DI, Reuso Excessivo de Strings, Adapter Flexível, Operações de IO, Herança e Listener Escondido.

4.3.1 Evite Imagens. Esta categoria reúne respostas que indicam como má prática o uso de imagens quando poderia ser usado um Drawable Resource em XML. De forma similar, respostas indicam como boas práticas o uso de Drawable Resources, criado com XML, sempre que possível. Exemplos de frases que indicaram más práticas são: P23 diz *"Uso de formatos não otimizados, uso de drawables onde recursos padrão do Android seriam preferíveis"*, P37 diz *usar jpg ou png para formas simples é ruim, apenas as desenhe [através de* Drawable Resources*]*. Exemplo de frase que indicou boa prática é P36 que diz *"[...] Quando possível, criar resources através de xml."*. Apenas o elemento Drawable Resoirce entrou nessa categoria.

4.3.2 Activity Inexistente. Esta categoria reúne respostas que indicam como má prática classes manterem referência a uma Activity, pois como ela possui ciclo de vida, quando a classe tentar acessá-la, a Activity pode não existir mais, resultando em possíveis erros na aplicação. De forma similar, respostas indicam como boas práticas, elementos com ciclo de vida independente, não manter referência à Activitys. Exemplos de frases que indicaram más práticas são: P28 diz *"Fazer Activities serem callbacks de processos assíncronos gerando memory leaks. Erros ao interpretar o ciclo de vida"*, P31 diz *"[...] ter referência estática para Activities, resultando em vazamento de memória"* (tradução livre). Exemplos de frases que indicaram boas prática são: P31 diz *"Não manter referências estáticas para Activities (ou classes annimas criadas dentro delas)"*, P4 diz *"Deus mata um cachorro toda vez que alguém passa o contexto da Activity para um componente que tem um ciclo de vida independente dela. Vaza memória e deixa todos tristes."*. Os elementos que entraram nessa categoria foram: Activitys e Listeners.

4.3.3 Reuso Excessivo de Strings. Esta categoria reúne respostas que indicam como má prática reutilizar o mesmo String Resource em muitos lugares no aplicativo, apenas porque o texto coincice, pois caso seja necessário alterar em um lugar, todos os outros serão afetados. De forma similar, respostas indicam como boas práticas considerar a semntica ou contexto ao nomear um String Resource, para mesmo que o valor seja o mesmo, os recursos sejam diferentes. Exemplos de frases que indicaram más práticas são: P32 diz *"Utilizar uma String pra mais de uma activity, pois se em algum momento, surja a necessidade de trocar em uma, vai afetar outra."*, P6 diz *"Reutilizar a string em várias telas"* (tradução livre) e P40 diz *"Reutilizar a string apenas porque o texto coincide, tenha cuidado com a semntica"* (tradução livre). Exemplos de frases que indicaram boas prática são: P32 diz *"Sempre busco separar em blocos, cada bloco representa uma activity e nunca aproveito uma String pra outra tela."* e P9 diz *"Não tenha medo de repetir strings [...]"*. Apenas o elemento String Resource entrou nessa categoria.

4.3.4 Listener Escondido. Esta categoria reúne respostas que indicam como má prática o uso de métodos de eventos do usuário no XML de layout, como por exemplo o método onClick. Respostas indicam como boas práticas o uso da biblioteca *ButterKnife* [25]. Exemplos de frases que indicaram más práticas são: P34 diz *"Nunca crie um listener dentro do XML. Isso esconde o listener de outros desenvolvedores e pode causar problemas até que ele seja encontrado"*, P39 e P41 expressam a mesma opinião, P41 ainda complementa dizendo que *"XML de layout deve lidar apenas com a view e não com ações"*. Exemplos de frases que indicaram boas prática são: P39 diz apenas *"Uso ButterKnife"*, P34 também expressa sua preferência por essa biblioteca. Apenas Listeners entraram nessa categoria.

5 DISCUSSÃO

[Under construction]

6 AMEAÇAS A VALIDADE

[Under construction]

7 CONCLUSÃO

[Under construction]

REFERENCES

[1] Activities. https://developer.android.com/guide/components/activities.html. (????). Last accessed at 29/08/2016.

[2] Android Studio. https://developer.android.com/studio/index.html. (????). Last accessed at 30/08/2016.

[3] Green Robot - Event Bus). http://greenrobot.org/eventbus. (????). Last accessed at 11/04/2017.

[4] GUI Architectures). https://www.martinfowler.com/eaaDev/uiArchs.html. (????). Last accessed at 11/04/2017.

[5] Wikipédia - Model-View-Presenter. https://pt.wikipedia.org/wiki/Model-view-presenter. (????). Last accessed at 11/04/2017.

[6] Wikipédia - Model-View-ViewModel. https://en.wikipedia.org/wiki/Modelviewviewmodel. (????). Last accessed at 11/04/2017.

[7] 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[8] 2016. PMD (2016). https://pmd.github.io. (2016). Last accessed at 29/08/2016.

[9] 2016. Wikipedia Code Smell. https://en.wikipedia.org/wiki/Code_smell. (2016). Last accessed at 14/11/2016.

[10] Domenico Amalfitano, Anna Fasolino, Porfirio Tramontana, Salvatore Carmine, and Atif Memon. 2012. Using GUI ripping for automated testing of Android applications. (2012).

[11] Maurcio Aniche and Marco Gerosa. 2016. Architectural Roles in Code Metric Assessment and Code Smell Detection. (2016).

[12] Lionel C Briand, William M Thomas, and Christopher J Hetmanski. 1993. Modeling and managing risk early in software development. In *Software Engineering, 1993. Proceedings., 15th International Conference on.* IEEE, 55–65.

[13] Erika Chin, Adrienne Felt, Kate Greenwood, and David Wagner. 2011. Analyzing inter-application communication in Android. (2011).

[14] Enck, William, and Patrick Drew McDaniel Machigar Ongtang. 2009. Understanding Android Security. (2009).

[15] Enck, William, and Patrick McDaniel Machigar Ongtang. 2008. Mitigating Android software misuse before it happens. (2008).

[16] Zheran Fang and Yingjiu Li Weili Han. 2014. Permission Based Android Security: Issues and Countermeasures. (2014).

[17] A. Milani Fard and A. Mesbah. 2013. JSNOSE: Detecting javascript code smells. (2013).

[18] Martin Fowler. 1999. *Refactoring. Improving the Design of Existing Code.* Addison-Wesley.

[19] Martin Fowler. 2006. Code Smell. http://martinfowler.com/bliki/CodeSmell.html. (Feb. 2006).

[20] Golnaz Gharachorlu. 2014. *Code Smells in Cascading Style Sheets: An Empirical Study and a Predictive Model.* Ph.D. Dissertation. The University of British Columbia.

[21] Marion Gottschalk, Mirco Josefiok, Jan Jelschen, and Andreas Winter. Removing Energy Code Smells with Reengineering Services. (????). Maus cheiros relacionados ao consumo de energia.

[22] Geoffrey Hecht. 2015. An Approach to Detect Android Antipatterns. (2015).

[23] Cuixiong Hu and Iulian Neamtiu. 2011. Automating GUI testing for Android applications. (2011).

[24] Arthur J. Riel. 1996. *Object-Oriented Design Heuristics.* Addison-Wesley Publishing Company. https://books.google.com.br/books?id=oHkhAQAAIAAJ

[25] Wharthon Jake. Butter Knife. http://jakewharton.github.io/butterknife/. (????). Last accessed at 11/04/2017.

[26] K Kavitha, P Salini, and V Ilamathy. 2016. Exploring the Malicious Android Applications and Reducing Risk using Static Analysis. (2016).

[27] Mario Linares-Vásquez, Sam Klock, Collin Mcmillan, Aminata Sabanè, Denys Poshyvanyk, and Yann-Gaël Guéhéneuc. Domain Matters: Bringing Further Evidence of the Relationships among Anti-patterns, Application Domains, and Quality-Related Metrics in Java Mobile Apps. (????).

[28] Umme Mannan, Danny Dig, Iftekhar Ahmed, Carlos Jensen, Rana Abdullah, and M Almurshed. Understanding Code Smells in Android Applications. (????). DOI:https://doi.org/10.1145/2897073.2897094

[29] Robert Martin. 8thlight Blog - The Clean Architecture. https://8thlight.com/blog/uncle-bob/2012/08/13/the-clean-architecture.html. (????). Last accessed at 11/04/2017.

[30] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship* (1 ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

[31] Nachiappan Nagappan and Thomas Ball. 2005. Static Analysis Tools As Early Indicators of Pre-release Defect Density. In *Proceedings of the 27th International Conference on Software Engineering (ICSE '05).* ACM, New York, NY, USA, 580–586. DOI:https://doi.org/10.1145/1062455.1062558

[32] Jan Reimann and Martin Brylski. 2013. A Tool-Supported Quality Smell Catalogue For Android Developers. (2013).

[33] Android Developer Site. Andriod Fundamentals. https://developer.android.com/guide/components/fundamentals.html. (????). Last accessed at 04/09/2016.

[34] Android Developers Site. Android RecyclerView. https://developer.android.com/training/material/lists-cards.html. (????). Last accessed at 12/04/2017.

[35] Android Developer Site. Optimizing View Hierarchies). https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies.html). (????). Last accessed at 09/04/2017.
[36] Android Developer Site. 2016. Building Your Firt App. https://developer.android.com/training/basics/firstapp/creating-project.html. (2016). Last accessed at 31/03/2017.
[37] Android Developer Site. 2016. Documentação Site Android Developer. https://developer.android.com. (2016). Last accessed at 27/10/2016.
[38] Developer Android Site. 2016. Resources Overview. https://developer.android.com/guide/topics/resources/overview.html. (2016). Last accessed at 08/09/2016.
[39] Nikolaos Tsantalis. 2010. *Evaluation and Improvement of Software Architecture: Identification of Design Problems in Object-Oriented Systems and Resolution through Refactorings*. Ph.D. Dissertation. University of Macedonia.
[40] Daniël Verloop. 2013. *Code Smells in the Mobile Applications Domain*. Ph.D. Dissertation. TU Delft, Delft University of Technology.
[41] Wikipédia. Inheritance (object-oriented programming). https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming). (????). Last accessed at 08/04/2017.
[42] Wenjia Wu, Jianan Wu, Yanhao Wang, and Ming Yang Zhen Ling. 2016. Eficient Fingerprinting-based Android Device Identication with Zero-permission Identiers. (2016).
[43] A. Yamashita and L. Moonen. 2012. Do code smells reflect important maintainability aspects?. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 306–315. DOI: https://doi.org/10.1109/ICSM.2012.6405287
[44] S. Yu. 2016. Big privacy: Challenges and opportunities of privacy study in the age of big data. (2016).
[45] Yuan Zhang, Min Yang, Zhemin Yang, Guofei GU, and Binyu Zang Peng Ning. 2004. Exploring Permission Induced Risk in AndroidApplications for Malicious Detection. (2004).
[46] Yuan Zhang, Min Yang, Zhemin Yang, and Binyu Zang. 2014. Permission Use Analysis for Vetting Undesirable Behaviors in Android Apps. (2014).