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

Suelen G. Carvalho
Universidade de São Paulo
Rua do Matão, 1010
So 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, Netherland 2628
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),* 7 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 escreverem código com qualidade incluindo *design patterns* e *code smells*. Defeitos de software, ou *bugs*, podem custar a empresas quantias significativas, especialmente quando conduzem a falhas de software [6, 21]. Evolução e manutenção de software também já se provaram como os maiores gastos com aplicações [28].

Code smells desempenham um importante papel na busca por qualidade de código. 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, Checkstyle e FindBugs são exemplos de ferramentas que identificam automaticamente alguns tipos de *code smells* em códigos Java.

Determinar o que é ou não um *code smells* é subjetivo e pode variar de acordo com tecnologia, desenvolvedor, metologia de desenvolvimento dentre outros aspectos []. Alguns estudos têm buscado por *code smells* tradicionais em projetos Android. Por exemplo, Verloop [29] analisou se classes derivadas do SDK Android são mais ou menos propensas a code smells tradicionais do que classes puramente Java. Linares et al. [18] usaram o método DECOR para realizar a detecção de 18 *anti-patterns* orientado a objetos em aplicativos móveis. Outros estudos identificaram *code smells* específicos Android, porém relacionados ao consumo inteligente de recursos do dispositivo, como bateria e memória, usabilidade, dentre outros [13, 22].

Nossa pesquisa complementa as anteriores no sentido de que também buscamos *code smells* Android, e se difere delas pois estamos buscando *smells* 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 o usuário. Buscamos responder questões como *"quais são as **boas e más** práticas ao lidar com* ACTIVITY*s,* FRAGMENT*s,* ADAPTER*s e* LISTENER*s?"* ou *"quais são as **boas e más** práticas para a construção da interface visual?"*. Respondemos a essas e outras questões com base na percepção de desenvolvedores dessa plataforma.

Optamos por focar em elementos relacionados ao *front-end* Android pois encontramos pesquisas com uma curiosidade similar, porém relacionadas a identificação de *smells* em tecnologias usadas no *front-end* de projetos web [5, 11, 12]. E enquanto que *smells* em projetos Java já foram extensivamente estudados [], o *front-end* Android possui peculiaridades não encontradas, e portanto não investigadas, em código Java tradicional. Podemos citar alguns exemplos como o ciclo de vida definido pela plataforma que é atribuído a toda ACTIVITY ou FRAGMENT e a criao da interface visual de ACTIVITYs

SBES'17, Fortaleza, Ceará Brasil

DOI: 10.475/123_4

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 [26] e chegamos nos seguintes itens: ACTIVITYS, FRAGMENTS, LISTENERS, ADAPTERS e os *applications resources*, que são arquivos XML ou imagens utilizados na interface visual como por exemplo DRAWABLES, LAYOUTS, STYLES e COLORS.

Como existem muitos tipos de *applications resources* [27], com o objetivo de limitar o tamanho do questionário, selecionamos quatro: LAYOUT, STYLES, STRING e DRAWABLE. Esses recursos estão presentes no template padrão do Android Studio [25], IDE oficial para desenvolvimento de projetos da plataforma [2].

Com isso, pretendemos responder as seguintes questões de pesquisa:

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

RQ2 Códigos afetados pelos *code smells* propostos são percebidos pelos desenvolvedores como problemáticos?

Para coletar os dados iniciais, publicamos um questionário online sobre boas e más práticas no *front-end* Android. Perguntas dissertativas foram usadas para não limitar o participante, possibilitar respostas mais completas e evitar enviesamento.

As seções seguintes deste artigo esto organizadas da seguinte forma: na Seção 2 discutimos alguns trabalhos relacionados e o estado da arte sobre Android e *code smells*. Na Seção 3 falamos sobre os métodos utilizados em nosso estudo. A Seção 4 apresenta os resultados e as ameaças à validade do nosso estudo. Na Seção 5 discutimos e concluímos.

2 TRABALHOS RELACIONADOS

Muitas pesquisas têm sido realizadas sobre a plataforma Android, muitas delas focam em vulnerabilidades [7–9, 17, 31, 34, 35], autenticação [10, 32, 33] e testes [4, 15]. 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 [29] 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 [13] 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 [23]. 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 [14, 19, 22], 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 [12] e JavaScript [11]. 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.

Tabela 2: Experiência dos participantes com desenvolvimento Android.

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 %

3.3 Análise dos Dados

O processo de análise partiu da listagem das 45 respostas do questionário e se deu em 5 passos: verticalização, limpeza dos dados, codificação, divisão e agrupamento das categorias.

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 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

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

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

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.*

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. Agrupamos essas categorias em 6 temas: Conceitos Tradicionais, Problemas Arquiteturais, Problemas Pontuais, Ferramentas Utilitárias, Conhecimento e Estrutura de Arquivos.

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.

A Tabela 3 apresenta o número de recorrências, em cada questão sobre boas e más práticas, das categorias de alta, média e baixa recorrência agrupadas por temas. A última linha da Tabela, #Categorias, apresenta quantas categorias emergiram de cada questão, ou seja, **com base em um elemento Android**, quais são os pontos de atenção (boas e más práticas) que devemos avaliar pensando em qualidade de código. Já a última coluna da Tabela, #Questões, apresenta em quantas questões cada categoria surgiu, ou seja, **com base na categoria** (que agrupa observações sobre boas e más práticas), quais elementos devem ser investigados pensando em qualdiade de código.

Esta seção está organizada em 6 subseções. As primeiras 5 subseções abordam as categorias apenas de alta, média e baixa recorrência dos temas: Conceitos Tradicionais, Problemas Arquiteturais, Problemas Pontuais, Ferramentas Utilitárias e Conhecimento (não há categorias com essas recorrências no grupo Estrutura de Arquivos). Por último temos uma subseção para falar apenas das categorias de baixíssima recorrência.

4.1 Categorias de Alta Recorrência

Obtivemos 4 categorias consideradas de alta recorrência: No Logic In View, Resource Name Pattern, Magic Resource e Nested Layout.

4.1.1 No Logic In View. Esta categoria reúne respostas que indicam como má prática haver regra de negócio nos elementos Android afetados. De forma similar, respostas indicam como boas práticas não haver código de regra de negócio. 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"*. Os elementos que entraram nessa categoria foram: Activitys, Fragments, Listeners e Adapters.

4.1.2 Resource Name Pattern. 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 indica um padrão similar, porém sufixo para 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"*, 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. P6 sugeriu a convenção `[screen]_[type]_[text]` e citou como exemplo `welcome_message_title` e `registration_field_name`. Não foram sugeridos nenhum padrão para Styles Resources e Drawable Resources.

4.1.3 Magic Resource. 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 foi inspirado no *code smell Magic Number* [20] 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.

Tabela 3: Lista de categorias de alta, média e baixa recorrência vs. ocorrência nas questões sobre boas e más práticas.

Categoria	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q14	Q15	Q16	Q17	Q18	#Questões
No Logic In View	12	15	6	8	3	8	–	–	–	–	–	–	–	–	–	–	–	–	6
Resource Name Pattern	1	–	–	–	–	–	–	–	3	2	3	2	8	2	3	–	–	–	8
Magic [Possible] Resource	–	–	–	–	–	–	–	–	4	2	1	1	9	6	–	–	–	–	6
Nested Layout	–	–	–	–	1	–	–	–	9	9	–	–	–	–	–	–	1	1	5
MVP	8	2	5	–	–	–	–	–	–	–	–	–	–	–	–	–	2	–	4
View Coupled To View	–	2	4	6	–	3	1	2	–	–	–	–	–	–	–	–	–	–	6
Life Cycle	4	3	3	5	–	–	1	–	–	–	–	–	–	–	–	–	–	–	5
Use Include	–	–	–	–	–	–	–	–	12	2	–	–	–	–	–	–	1	–	3
Use View Holder Pattern	–	–	–	–	12	2	–	–	–	–	–	–	–	–	–	–	–	–	2
Drawable Size Matters	–	–	–	–	–	–	–	–	1	1	–	–	–	–	4	6	–	–	4
Suspicious Extra Knowledge About Behavior	1	–	–	–	1	3	5	2	–	–	–	–	–	–	–	–	–	–	5
God/Large Class *	1	4	1	2	–	1	–	1	–	–	–	1	–	–	–	–	–	–	7
Use Fragment	3	2	5	1	–	–	–	–	–	–	–	–	–	–	–	–	–	–	4
Use Vector Drawables	–	–	–	–	–	–	–	–	–	–	–	–	–	–	11	–	–	–	1
Use Well Know Architectures	–	–	2	–	2	–	–	–	–	–	–	–	–	–	–	–	5	1	4
Use Fragment Just If Needed	–	–	8	2	–	–	–	–	–	–	–	–	–	–	–	–	–	–	2
God Style Resource	–	–	–	–	–	–	–	–	–	–	5	3	–	–	–	–	–	–	2
Messy String Resources	–	–	–	–	–	–	–	–	–	–	–	–	4	4	–	–	–	–	2
Avoid Images	–	–	–	–	–	–	–	–	1	–	–	–	–	–	4	2	–	–	3
Duplicated Styles Attributes	–	–	–	–	–	–	–	–	1	2	2	1	–	–	–	–	1	–	5
Zumbi Referended Activity	2	4	–	–	–	–	–	1	–	–	–	–	–	–	–	–	–	–	3
DI	1	–	–	–	–	–	4	–	–	–	–	–	–	–	–	–	1	–	3
Reuse String Resources Too Much	–	–	–	–	–	–	–	–	–	–	–	–	2	4	–	–	–	–	2
Flex Adapter	–	–	–	–	3	2	–	–	–	1	–	–	–	–	–	–	–	–	3
Inheritance **	2	–	2	–	1	–	–	–	–	–	–	–	–	–	–	–	–	–	3
Hided Listener	–	–	–	–	–	–	–	3	–	–	–	–	–	–	–	–	–	–	1
Opened Activity	–	2	–	–	–	–	–	1	–	–	–	–	–	–	–	–	–	–	2
#Categorias	10	8	9	6	7	6	4	6	7	7	4	5	4	4	4	2	6	2	

* *God Classe* [16] e *Large Class* [3] são *code smells* tradicionais previamente definidos em literaturas.

** *Inheritance*, ou herança, é um conceito da Programação Orientada a Objetos [30].

4.1.4 Nested Layout. 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 categoria. O site oficial do Android conta com informações e ferramentas automatizadas para lidar com esse sintoma [24].

4.2 Categorias de Média Recorrência

Obtivemos 15 categorias consideradas de média recorrência: View Coupled To View, MVP, Life Cycle, Use Include, Use View Holder Pattern, Suspicious Extra Knowledge About Behavior, Drawable Size Matters, God/Large Class, Use Fragment, Use Vector Drawables, Use Well Know Architectures, Use Fragment Just If Needed, God Style Resource, Messy String Resources e Duplicated Styles Attributes.

4.3 Categorias de Baixa Recorrência

Obtivemos 8 categorias consideradas de baixa recorrência: Avoid Images, Zumbi Referended Activity, DI, Reuse String Resources Too Much, Flex Adapter, IO Operations, Inheritance e Hided Listener.

4.4 Categorias de Baixíssima Recorrência

Obtivemos 29 categorias consideradas de baixíssima recorrência: MVC, Clean Architecture, Don't Use Fragment, Nested Fragment, Mix String Resources With Business Logic, Use 9 Patch Files, Dealing With App Stack Manually, Styles Knows Too Much, Package Structure, MVVM, Activity Handle More Than One Layout, Bad Relative, Deprecated Attributes, Fat

onCreate, HTML Into String File, Inherit From Support Library Always, Listener Has A Valid Context, Safe Adapter, Single Activity, Static Things On Adapter, Style Into String File, Use Constraint Layout, Dead Resources, DRY, SOLID, Reuse, Singleton, Opened Activity e Unnecessary ViewGroup

5 DISCUSSÃO

[Under construction]

Consideramos inconclusivo se a utlização de Fragments é recomendada ou não. 2 participantes afirmaram enfaticamente que não se deve usar FRAGMENTS, por exemplo, segundo P7, uma má prática sobre FRAGMENTS é "o uso deles".

6 AMEAÇAS A VALIDADE

[Under construction]

Consideramos inconclusivo se a utlização de Fragments é recomendada ou não. 2 participantes afirmaram enfaticamente que não se deve usar FRAGMENTS, por exemplo, segundo P7, uma má prática sobre FRAGMENTS é "o uso deles".

7 CONCLUSÃO

[Under construction]

Consideramos inconclusivo se a utlização de Fragments é recomendada ou não. 2 participantes afirmaram enfaticamente que não se deve usar FRAGMENTS, por exemplo, segundo P7, uma má prática sobre FRAGMENTS é "o uso deles".

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

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

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

[6] 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.

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

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

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

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

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

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

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

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

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

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

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

[18] 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. (????).

[19] 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

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

[21] 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

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

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

[24] Android Developer Site. Optimizing View Hierarchies). https://developer.android.com/topic/performance/rendering/optimizing-view-hierarchies.html). (????). Last accessed at 09/04/2017.

[25] 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.

[26] Android Developer Site. 2016. Documentação Site Android Developer. https://developer.android.com. (2016). Last accessed at 27/10/2016.

[27] Developer Android Site. 2016. Resources Overview. https://developer.android.com/guide/topics/resources/overview.html. (2016). Last accessed at 08/09/2016.

[28] 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.

[29] Daniël Verloop. 2013. *Code Smells in the Mobile Applications Domain*. Ph.D. Dissertation. TU Delft, Delft University of Technology.

[30] Wikipédia. Inheritance (object-oriented programming). https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming). (????). Last accessed at 08/04/2017.

[31] Wenjia Wu, Jianan Wu, Yanhao Wang, and Ming Yang Zhen Ling. 2016. Eficient Fingerprinting-based Android Device Identication with Zero-permission Identiers. (2016).

[32] 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

[33] S. Yu. 2016. Big privacy: Challenges and opportunities of privacy study in the age of big data. (2016).

[34] Yuan Zhang, Min Yang, Zhemin Yang, Guofei GU, and Binyu Zang Peng Ning. 2004. Exploring Permission Induced Risk in AndroidApplications for Malicious Detection. (2004).

[35] Yuan Zhang, Min Yang, Zhemin Yang, and Binyu Zang. 2014. Permission Use Analysis for Vetting Undesirable Behaviors in Android Apps. (2014).