Cheiros de Código em projetos Android. Um estudo qualitativo sobre a percepção de desenvolvedores

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

Marco Aurélio Gerosa
Universidade de São Paulo
Rua do Matão, 1010
São Paulo, SP 05508-090
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 android 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, validado 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. Cheiros de Código em projetos Android. Um estudo qualitativo sobre a percepção de desenvolvedores. In *Proceedings of SBES 2017: 31st Brazilian Symposium on Software Engineering, Fortaleza, Ceará Brasil, Setembro 2017 (SBES'17),* 3 pages.
DOI: 10.475/123_4

1 INTRODUÇÃO

Escrever código com qualidade tem se tornado cada vez mais importante com o advento 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 de dinheiro, especialmente quando conduzem a falhas de software [2, 10]. Evolução e manutenção de software também já se provaram como os maiores gastos com aplicações [11].

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 forma automática no código. PMD, Checkstyle e FindBugs por exemplo, são ferramentas que identificam automaticamente por 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 tem buscado por *code smells* tradicionais em projetos android, outros estudos tem buscado por *code smells* relacionados ao consumo inteligente de recursos de dispositivos android. Há estudos que avaliam quais *code smells* aparecem com maior frequência em aplicações android, tradicionais ou algum específico à plataforma.

Apesar de já existirem diversas pesquisas sobre *code smells* em projetos android, nenhuma delas responde questões como quais são as más práticas ao lidar com `android resources`, ou ao lidar com `activity`s, `fragment`s, `adapter`s e `listener`s. Nossa pesquisa objetiva definir *code smells* android baseado em o que desenvolvedores desta plataforma percebem como boas e más práticas em elementos específicos da plataforma.

Nas seções seguintes deste artigo, discutiremos primeiro alguns trabalhos relacionados (Seção 2) e os métodos utilizados em nosso estudo (Seção 3). A Seção 4 apresenta os resultados e as ameaças à validade do nosso estudo. Na Seção 5 discutimos e concluímos... e terminamos com uma discussão de trabalhos futuros (Seção 6).

2 TRABALHOS RELACIONADOS

Muitas pesquisas têm sido realizadas sobre a plataforma android, muitas delas focam em vulnerabilidades [3–5, 9, 13, 16, 17], autenticação [6, 14, 15] e testes [1, 8]. Diferentemente destas 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 code smells, podemos chegar a heurísticas para identificá-los e com estas heurísticas,

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SBES'17, Fortaleza, Ceará Brasil
© 2017 Copyright held by the owner/author(s). 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

implementar ferramentas que automatizem o processo de identificar códigos maus cheirosos.

Verloop conduziu um estudo onde 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 [12]. Nossa pesquisa se relaciona com a de Verloop no sentido de que também estamos buscando por cheiros de código, entretanto, ao invés de buscarmos por cheiros de código já definidos, realizamos uma abordagem inversa onde, 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. conduzem um estudo sobre formas de detectar e refatorar cheiros de código relacionados a uso eficiênte de energia [7]. Os autores compilam um catálogo com 8 cheiros de código e trabalham 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 a 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.

3 METODOLOGIA

Para obter as ideias iniciais para esta pesquisa nós publicamos um survey online com perguntas dissertativas sobre boas e más práticas em determinados elementos da plataforma android. O objetivo das perguntas dissertativas foi obter o máximo de informações dos desenvolvedores. O survey foi escrito em inglês porém era informado que tanto respotas inglês ou português eram aceitas. Antes da divulgação, validamos o survey com 3 desenvolvedores android, com o feedback deles fizemos alguns ajustes relacionados a obrigatoriedade de algumas respostas.

O survey foi dividido em 3 sessões. A primeira continha perguntas para mapeamento demográfico e de perfil. A segunda continham perguntas relacionadas s boas e más práticas em elementos da plataforma android. A terceira e última sessão tentava obter alguma ideia não coletada na sessão anterior além do email do participante caso o mesmo se disponibilizasse a participar de outras etapa da pesquisa. Mais detalhes sobre a criação do questionário será abordado da Sessão 3.1. Após coletada as respostas, realizamos uma análise qualitativa. Essa análise é é detalhada na sessão 3.2.

3.1 Questionário

O survey foi escrito em inglês e continha uma mensagem informando que respotas em inglês ou português eram aceitas. Criamos e modificamos o survey coletando respostas experimentais; qualquer alteração feita no questionário foi baseada nas respostas que recebemos de nossos 3 participantes iniciais. Realizou-se a análise qualitativa nas respostas do survey. Esse processo é discutido em detalhes na Seção III-F.

3.2 Participantes

Este survey foi respondido por 45 participantes. O survey foi divulgado abertamente em redes sociais. Coletamos 36 respostas do Brasil, 7 da Europa, 1 dos Estados Unidos e 1 preferiu não responder. Quarenta dos participantes tinham 3 anos ou mais de 10 anos de experiência com desenvolvimento, destes, trinta e um participantes tinham de 3 a 8 anos de experiência com android. Trinta e três dos participantes tinham de 25 a 34 anos, 8 de 19 a 24 e 3 de 35 a 54.

3.3 Categorização e Identificação dos Smells

A análise partiu da listagem das 45 respostas ao questionário. A partir desta listagem realizamos o processo que denominamos como verticalização onde foi possvel extrair 810 respostas sobre boas e más práticas em 8 tipos de elementos android (Activity, Fragment, Adapter, Listener, layout, string, style e drawable resources) e boas e más práticas gerais em android. O número 810 refere-se a 8 elementos android x 2 perguntas (uma questionando sobre boas práticas e outra sobre más práticas) + 2 perguntas genéricas sobre boas e más práticas, ou seja: 18 perguntas sobre boas e más práticas respondidas por 45 participantes, verticalizando, ou seja, cada resposta de boa ou má prática tornando-se um registro a ser analisado, resulta em 810 respostas sobre boas ou más práticas.

Nosso primeiro passo foi realizar a limpeza dos dados. Esta etapa consistiu em remover respostas obviamente não úteis como as que constavam em branco, continham frases como "Não", "Não que eu saiba", "I don't remember" 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 "Like all Java code, the overkill of Utils...". Foi criada uma coluna USABLE onde era informado "yes" ou "no". 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.

Após a verticalização, foram realizadas diversas iterações nas respostas consideradas a fim de categorizá-las em algum smell pré-existente (já catalogado) ou em algum android smell. Essas iterações consistiam em analisar resposta a resposta da lista e atribuir 1 ou mais categorias de algum possvel smell (pré-existente ou android smell). Foram realizadas diversas iterações de categorização com o objetivo de normalizar as categorias, ou seja, evitar que categorias que apontem para padrões similares recebam nomes diferentes, como por exemplo KAS (Keep Activity Simple) e KFS (Keep Fragment Simple) foram normalizadas para KXS (Keep Elemento Simple). Durante a análise houveram 30 respostas que não eram triviais de identificar uma categoria ou mesmo de dizer se estas respostas deveriam ser consideradas. Essas respostas foram marcadas como maybe durante o processo. 9 respostas foram trocadas para "no". Foi criada a coluna WHY

MAYBE OR NOT USABLE para indicar o motivo do "no" ou "maybe". Ao final, totalizavam em 313 respostas de fato consideradas, ou seja, marcadas com "yes" , e categorizadas.

Uma prática que consideramos importante para a normalização dos smells/categorias foi durante as iterações de categorização, cada categoria atribuda era adicionada a uma lista de categorias e era inserido descrições próximas as respostas dadas que indicava que tipo de respostas estavam entrando naquela categoria. Esta prática eliminou a criação de categorias com propósitos similares e nomes diferentes e serviu como base para a definição e avaliação da relevncia dos smells escolhidos para serem usados nos próximos passos.

As respostas que apresentavam mais de uma categoria foram quebradas em duas ou mais linhas de respostas, de acordo com o número de categorias identificadas, de forma a ser possvel contabilizar a incidência. Por exemplo, a resposta "Não fazer Activities serem callbacks de execuções assncronas. Herdar sempre das classes fornecidas pelas bibliotecas de suporte, nunca diretamente da plataforma" indica na primeira oração a categoria Zumbi Referended Activity e na segunda, a categoria Inherit From Support Library Always, desta forma, ao separar foi mantido o texto (coluna opinion) apenas relativo a categoria, como se fossem duas respostas válidas. Algumas respostas não estava tão bem separadas qual frase indicava qual categoria, ou as vezes a resposta completa era necessário para entender ambas as categorizações, nestes casos, preferiu-se manter a resposta original, mesmo que duplicada, e categorizar cada uma de forma diferente. Após estas divisões, as 313 respostas iniciais se tornaram 388.

Como última etapa do processo de categorização e identificação dos smells, repassamos por todas as 30 respostas marcadas como maybe e foi decidido se seria usada ou não. 6 delas se tornaram "yes" e foram atribudas categorias, 24 se tornaram "no" e o motivo foi indicado na coluna WHY MAYBE OR NOT USABLE. Ao final, a planilha de análise contém respostas marcadas como úteis ou não e nenhuma marcada como "maybe". Ao final, totalizamos 319 respostas consideradas, considerando as divisões realizadas conforme mencionado acima, totalizamos com 394 respostas sobre boas e más práticas categorizadas. Durante esta etapa final, um caso interessante é que foram identificadas diversas respostas indicando que não se deve usar Fragments porém sem grande argumentações sobre o motivo, por exemplo: "Fragments are the spawn of satan" e "I try to avoid them". Estas respostas foram consideradas pela quantidade derepetições em que apareceram.

Em resumo, o processo de categorização e identificação dos smells seguiu as seguintes etapas:

Limpeza dos Dados ¿ Verticalização ¿ Iterações de Categorização ¿ Divisões ¿ Eliminação das Dúvidas (respostas marcadas com maybe)

Ao final desta etapa, conclumos com uma lista com 46 categorias de smells. Para filtrarmos quais smells seriam usados nas próximas etapas. Nosso objetivo neste momento era entender quais são as categorias mais notadas pelos desenvolvedores que ale a pena refirnar e validar? Avaliamos a relevncia contabilizando em quantas boas e más práticas ele foi utilizando. Esta contabilização resultou em

4 RESULTADOS

Under construction.

5 DISCUSSÃO

Under construction

6 TRABALHOS FUTUROS

Under construction

7 CONCLUSÃO

Under construction

REFERENCES

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

[2] 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.

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

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

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

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

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

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

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

[10] 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

[11] 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.

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

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

[14] 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

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

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

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