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),* 2 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.

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, 12, 15, 16], autenticação [6, 13, 14] 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,

SBES'17, Fortaleza, Ceará Brasil

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 [11]. 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

Under construction.

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

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

[13] 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

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

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

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