Um Conjunto Validado de Maus Cheiros na Camada de Apresentação de Aplicativos Android

Suelen Goularte Carvalho

Dissertação apresentada
ao
Instituto de Matemática e Estatística
da
Universidade de São Paulo
para
obtenção do título
de
Mestre em Ciências

Programa: Mestrado em Ciência da Computação
Orientador: Marco Aurélio Gerosa, Ph.D.

São Paulo, Julho de 2016

Um Conjunto Validado de
Maus Cheiros na Camada de Apresentação
de Aplicativos Android

Esta é a versão original da dissertação elaborada
pelo candidato Suelen Goularte Carvalho, tal como
submetida a Comissão Julgadora.

Comissão Julgadora:

- Marco Aurélio Gerosa, Ph.d. – IME-USP

Dedico esta dissertação de mestrado a minha mãe.

"O motivo do tempo é que tudo não acontece de uma vez só."
– Albert Einstein

Agradecimentos

A fazer.

Resumo

Uma tarefa que constantemente desenvolvedores de software precisam fazer é identificar trechos de código problemáticos para que possam refatorar, sendo o objetivo final ter constantemente uma base de código fácil de ser mantida e evoluída. Para isso, os desenvolvedores costumam fazer uso de estratégias de detecção de maus cheiros de código (*code smells*). Apesar de já existirem diversos maus cheiros catalogados, como por exemplo *God Class*, *Long Method*, dentre outros, eles não levam em consideração a natureza do projeto. Entretanto, projetos Android possuem particularidades relevantes e não experimentadas até o momento, por exemplo o diretório `res` que armazena todos os recursos usados na aplicação ou, uma ACTIVITY que por natureza, acumula diversas responsabilidades. Pesquisas nesse sentido, específicas sobre projetos Android, ainda estão em sua infância. Nesta tese pretendemos identificar, validar e documentar maus cheiros de código Android com relação a camada de apresentação, onde se encontra grandes distinções quando se comparado a projetos tradicionais. Em outros trabalhos sobre maus cheiros Android, foram identificados maus cheiros relacionados a segurança, consumo inteligente de recursos ou que de alguma maneira influênciavam a experiência ou expectativa do usuário. Diferentemente deles, nossa proposta é catalogar maus cheiros Android que influênciem a qualidade do código. Com isso os desenvolvedores terão mais uma ferramenta aliada para a produção de código de qualidade.

Palavras-chave: android, maus cheiros, qualidade de código, engenharia de software, manutenção de software, métricas de código.

Abstract

Keywords: android, software engineering, code smells, software maintance, code metrics, code quality.

Sumário

Lista de Abreviaturas

SDK *Software Development Kit*

IDE *Integrated Development Environment*

APK *Android Package*

ART *Android RunTime*

Lista de Símbolos

Σ Sistema de transição de estados

Lista de Figuras

Capítulo 1

Introdução

No próximo ano o Android completará uma década desde seu primeiro lançamento em 2007 e não há dúvidas sobre o seu sucesso [App16a, Per16, Sma16]. Atualmente há disponível mais de 2 milhões de aplicativos Google Play Store, loja oficial de aplicativos Android [App16b]. Mais de 83,5% dos *devices* no mundo usam o sistema operacional móvel Android (plataforma Android), e este percentual vem crescendo ano após ano [Gro16, Mob16]. Hoje podemos encontrá-lo em outros dispositivos como *smart TVs*, *smartphones*, carros, dentre outros [And16, And14]. Estes dados somado a experiência do autor são os motivadores por esta tese se tratar desta plataforma.

Duas tarefas que desenvolvedores de software constantemente fazem é criar código novo e alterar código existente, em ambos os casos busca-se escrever código de qualidade. Para a primeira, criar código novo, desenvolvedores costumam se apoiar em boas práticas e *design patterns* já estabelecidos [ACM03, RW12, GHJV94]. Para a segunda, alterar código existente, um dos motivos pode ser refatorar, ou seja, tornar melhor o código sem alterar o comportamento [Ref99]. Para isso, é comum se utilizar de estratégias de detecção de maus cheiros de código (*code smells*), estes apontam trechos de códigos que podem se beneficiar de refatoração. Apesar de já existir um catálogo extenso de *code smells* a maioria deles não considera a natureza do projeto e suas particularidades, muitos deles foram elaborados pensando em projetos orientados a objetos (projetos tradicionais).

De fato projetos Android são orientado a objetos, no entanto possuem particularidades que ainda não foram experimentadas em projetos tradicionais, principalmente com relação a camada de apresentação, onde ele apresenta maiores distinções com relação a projetos tradicionais. Segundo [Hec15] *"Android-specific antipatterns are far more frequent and common than OO antipatterns."*. Por exemplo, além de código Java, grande parte de um projeto

Android é constituído por arquivos XML, que são os recursos da aplicação localizados dentro do diretório `res`, normalmente responsáveis por apresentar algo ao usuário como uma tela, uma imagem, uma tradução e assim por diante. No início do projeto estes arquivos costumam ser poucos e pequenos. Conforme o projeto evolui, esta quantidade aumenta junto com sua complexidade, trazendo problemas em encontrá-las, reaproveitá-las e entendê-las. Enquanto estes problemas já estão bem resolvidos em projetos tradicionais, ainda não há uma forma sistemática de identificá-los em recursos de projetos Android para que possam ser refatorados.

Outra particularidade é sobre componentes como ACTIVITIES que por natureza são classes que possuem muitas responsabilidades [Ver13], sempre estão vinculadas a um LAYOUT e normalmente precisam de acesso a classes do modelo da aplicação. Analogamente ao padrão MVC, ACTIVITIES fazem um pouco o papel de VIEW e CONTROLLER. Poderíamos classificá-la com o conhecido *code smell God Class* [Rie96], no entanto, a proposta de refatoração dada a ele não necessáriamente é a mais apropriada para uma ACTIVITY, que é um componente específico da plataforma Android, responsável pela apresentação e interações do usuário com uma tela [Anda].

Na prática, desenvolvedores Android percebem estes problemas diariamente há quase uma década. Muitos deles já se utilizam de práticas para solucioná-los, esta afirmação é enfatizada em [RB13] ao dizer que *"The problem in mobile development is that developers are aware of quality smells only indirectly because their definitions are informal (best-practices, bug tracker issues, forum discussions etc.) and resources where to find them are distributed over the web."*. Ou seja, não existe hoje um catálogo destas boas e más práticas, tornando difícil a detecção e sugestão de refatorações apropriadas as particularidades da plataforma.

No período de 2008 a 2015, nas principais conferências de manutenção de software, apenas 5 *papers* foram sobre maus cheiros Android, dentro de um total de 52 *papers* sobre o assunto [MDA+]. Apesar do rápido crescimento da plataforma Android, pesquisas em torno dela não veem sendo realizadas na mesma velocidade e a ausência de um catálogo de maus cheiros Android resulta em (i) uma carência de conhecimento sobre boas e más práticas a ser compatilhado entre praticantes da plataforma, (ii) indisponibilidade de ter uma ferramenta de detecção de maus cheiros de forma a alertar automaticamente os desenvolvedores da existência dos mesmos e (iii) ausência de estudo empírico sobre o ímpacto destas más práticas na manutenabilidade do código de projetos Android. Por este motivos, boas e más práticas que são específicos a uma platadforma, no caso Android, tem emergido como tópicos de pesquisa sobre manutenção de código [AGC+16].

1.1 Questões de Pesquisa

A questão primária desta pesquisa é **Quais são as boas e más práticas específicas à camada de apresentação em aplicações Android nativas?**. Para responder a esta questão, temos as seguintes questões secundárias:

- **Q1: Existe maus cheiros específicos para a camada de apresentação do Android?**

  Pretende-se coletar, através de um questionário, boas e más práticas utilziadas e/ou percebidas por desenvolvedores Android em cada elemento da camada de apresentação. Entendemos que os elementos que compoem a camada de apresentação de projetos Android são:

  - Classes que extendem `android.view.View`,
  - Classes que extendem `android.widget.Adapter`,
  - Classes que extendem `android.app.Fragment` ou a mesma representada dentro do pacote *Support Library*,
  - Classes que extendem `android.app.Activity` ou a mesma representada dentro do pacote *Support Library*,
  - Todo *listener*. Apesar de Android *listeners* extenderem a class `android.view.View`, pretende-se avaliá-la como um elemento separado pois no dia-a-dia do desenvolvimento ela desempenha um papel importante,
  - Todo conteúdo do diretório `res`. Cada tipo de recurso será considerado um elemento separado no questionário.

  Para cada elemento listado acima, pretende-se fazer as seguintes perguntas ("X" indica o elemento em pauta na questão):

  - Você conhece ou utiliza alguma boa prática para lidar com X?
  - Você considera algo uma má prática para lidar com X?

  Para eliminar o risco de não estarmos considerando algum outro elemento, percebido como parte da camada de apresentação, ao final teremos uma pergunta adicional:

  - Na sua opinião, existe algum elemento Android não questionado, que você considera fazer parte da camada de apresentação do aplicativo? Em havendo, poderia listar boas e más práticas para cada elemento sugerido?

Estes dados serão analisados, categorizados e extraídos maus cheiros. Estes maus cheiros serão validados com especialistas em desenvolvimento Android.

- **Q2: Desenvolvedores percebem as classes afetadas pelos maus cheiros propostos como problemáticas?**

  Após a definição de um catálogo de maus cheiros, pretende-se validar se desenvolvedores os percebem de fato como indicativos de trechos de códigos problemáticos. Para isso pretende-se conduzir um experimento. [Completar aqui]

  Esta questão contempla duas subquestões sobre a relação dos maus cheiros e a tendência a mudanças e introdução de *bugs* no projeto.

  - **Q2.1: Qual a relação entre os maus cheiros propostos e o tendência a mudanças do código?**
  - **Q2.2: Qual a relação entre os maus cheiros propostos e o tendência a introdução de *bugs* no código?**

1.2 Trabalhos Relacionados

Diversas pesquisas em torno de maus cheiros de código veem sendo realizadas ao longo dos últimos anos. Já existem inclusive diversos maus cheiros mapeados, porém poucos deles são específicos da plataforma Android [MDA+]. Segundo [Hec15] estudos sobre maus cheiros de código sobre aplicações Android ainda estão em sua infância, porém ainda assim são muito relevantes inclusive notando-se que é mais comum identificar em aplicativos Android maus cheiros mapeados exclusivamente para esta plataforma do que os maus cheiros tradicionais [LVKM+, Hec15].

Code Smells in the Mobile Applications Domain

O trabalho [Ver13] avalia a presença de *code smells* definidos por [Fow99] e [ML13] em projetos Android. Apesar das relevantes contribuições feitas, a conclusão sobre a incidência de tais *code smells*, não é plenamente conclusiva, visto que dos 6 *code smells* analisados (*Large Class*, *Long Method*, *Long Parameter List*, *Type Checking*, *Feature Envy* e *Dead Code*) apenas dois deles, *Long Method* e *Type Checking*, se apresentam com maior destaque (duas vezes mais provável) em projetos Android. Os demais apresentam uma diferença mínima em classes Android quando se comparados a classes não específicas do Android. Por fim, acaba por não ser conclusivo quanto a maior relevância deles em Android ou não.

[Ver13] conclui com algumas recomendações de refatoração de forma a mitigar a presença do *code smell Long Method*. Estas recomendações são o uso do atualmente já reconhecido padrão *ViewHolder* em classes do tipo `Adapters`. Ele também sugere um *ActivityViewHolder* de forma a extrair código do método `onCreate` e deixá-lo menor. Sugere também o uso do atributo `onClick` em XMLs de LAYOUT e MENU.

Esta tese se difere do trabalho [Ver13] pois pretende-se identificar, validar e catalogar, com base na experiência de desenvolvedores, quais boas e más práticas, ao longo da última década, eles vem praticando ou evitando, respectivamente em projetos Android. Ou seja, não pretende-se a partir de um catálogo já estabelecido de *code smells* e sim catalogar *code smells* específicos da camada de apresentação Android.

A Tool-Supported Quality Smell Catalogue For Android Developers

Outro trabalho muito relevante realizado neste tema é o [RB13] que, baseado na documentação do Android, documenta 30 *quality smells* específicos Android. No texto *quality smells* são definidos como *"A quality smell is a certain structure in a model, indicating that it negatively influences specific quality requirements, which can be resolved by particular model refactorings"*. Estes requisitos de qualidade são centrados no usuários (estabilidade, tempo de início, conformidade com usuário, experiência do usuário e acessibilidade), consumo inteligente de recursos (eficiência geral, no uso de energia e meméria) e segurança.

Esta tese se difere do trabalho [RB13] pois pretendemos encontrar *code smells* em termos de qualidade de código, ou seja, que influênciem na legibilidade e manutenabilidade do código do projeto.

1.3 Contribuições

Ao final de nossa pesquisa pretendemos contribuir com um catálogo validade de maus cheiros na camada de apresentação de aplicativos Android. Pretende-se também implementá-los na ferramenta de detecção automática de maus cheiros Páprika, de forma a desenvolvedores conseguirem de forma rápida identificar pontos específicos de projetos Android, a refatorar. Garantindo um código de qualidade, fácil de ser mantido e evoluído.

1.4 Organização da Tese

Este trabalho está organizados da seguinte forma:

- **Capítulo 1** Introdução

  Neste capítulo introduzimos o contexto atual do desenvolvimento de aplicativos Android. Apresentamos quais nossas motivações e o problema a ser resolvido. Damos também uma breve introdução sobre como pretendemos resolvê-lo.

- **Capítulo 2** Fundamentação Teórica

  Neste capítulo iremos munir o leitor de informações básicas relevantes para o entedimento do trabalho. Os assuntos aprofundados aqui são: Android, *Code Smells* e Qualidade de Código.

- **Capítulo 3** Pesquisa

  Neste capítulo apresentamos em detalhes toda as etapas da pesquisa. Quais tipos de coleta de dados foram usados. Quais os critérios de escolha dos respondentes e porquê. Como foram realizadas as análises dos dados e quais resultados obtivemos.

- **Capítulo 4** Catálogo de *Code Smells*

  Neste capítulo, catalogamos os *code smells* identificados.

- **Capítulo 5** Conclusão

  Neste capítulo apresentamos as conclusões do trabalho bem como as nossas limitações e trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Para a compreensão deste trabalho é importante ter claro a definição de 2 itens, são eles: *Aplicações Android* e *Maus Cheiros de Código*.

2.1 Android

Android é um sistema operacional móvel de código aberto, baseado no kernel do Linux, desenvolvido pelo Google para dispositivos móveis como *smartphones* e *tablets*. Atualmente é possível encontrá-lo sendo executado em outros dispositivos como painéis de carros, *smart TVs* e *smartwatchs*.

O primeiro *smartphone* com Android foi lançado em 2008 chamado G1, ou *Google One*. Desde seu lançamento o Android vem registrando um crescimento extremamente acentuado. Atualmente é a plataforma móvel mais utilizada no mundo, com 84% do *marketshare* [Mob16].

Aplicativos Android são arquivos do tipo `apk` que podem ser obtidos a partir de lojas de aplicativos virtuais. A *Google Play Store*, loja virtual oficial do Android, registra em 2016 mais de 2 milhões de aplicativos disponíveis e mais de 200 bilhões de downloads [App16b] [App16a].

Aplicativos Android

Aplicativos Android (apps) são desenvolvidos utilizando a linguagem Java e o Android SDK [Wika] e possuem recursos como imagens e arquivos `XML`. São executados sob uma máquina virtual, Dalvik ou ART, a depender da versão do Android em execução, sendo a última padrão nas versões mais recentes [Wikb] [Andb]. A IDE oficial para desenvolvimento é o Android Studio [Andc].

Um app consiste de um único arquivo com extensão `apk` contendo o código fonte compilado, recursos da aplicação, arquivos de dados e um arquivo de manifesto. Um projeto Android tem a seguinte estrutura:

- **Diretório** `java` contendo o código-fonte Java.
- **Diretório** `res`, derivado da palavra *resources*, contém todos os recursos como imagens, layout de telas e outros.
- **Arquivo** `AndroidManifest.xml` que contém configurações do projeto como: tela inicial, permissões de acesso a recursos do dispositivo e outros.

Componentes da Aplicação

Recursos

2.2 Qualidade de Código

2.3 Maus Cheiros

Mau cheiro de código é uma indicação superficial que usualmente corresponde a um problema mais profundo em um software. Por si só um *code smell*, seu termo em inglês, não é algo ruim, ocorre que frequentemente ele indica um problema mas não necessáriamente é o problema em si [Fow06]. O termo em inglês *code smell* foi cunhado pela primeira vez por Kent Beck enquanto ajudava Martin Fowler com o seu livro Refactoring [Fow99] [Fow06].

Code Smells são padrões de código que estão associados com um design ruim e más práticas de programação. Diferentemente de erros de código eles não resultam em comportamentos erroneos. *Code Smells* apontam para áreas na aplicação que podem se beneficiar de refatorações. [Ver13]. Refatoração é definido por "uma técnica para reestruturação de um código existente, alterando sua estrutura interna sem alterar seu comportamento externo" [Fow99].

Escolher não resolver *code smells* pela refatoração não resultará na aplicação falhar mas irá aumentar a dificuldade de mantê-la. Logo, a refatoração ajuda a melhorar a manutenabilidade de uma aplicação [Ver13]. Uma vez que os custos com manutenção são a maior parte dos custos envolvidos no ciclo de desenvolvimento de software [Tsa10], aumentar a manutenabilidade através de refatoração irá reduzir os custos de um software no longo prazo.

Capítulo 3

Pesquisa

3.1 Hipóteses

A fazer.

3.2 Processo de Pesquisa

A fazer.

3.2.1 Coleta de Dados

A fazer.

3.2.2 Análise dos Dados

A fazer.

3.3 Escrita dos Code Smells

A fazer.

Capítulo 4

Catálogo de Maus Cheiros

4.1 Code Smell 1

A fazer.

Capítulo 5

Conclusão

A fazer.

5.1 Principais contribuições

A fazer.

5.2 Trabalhos futuros

A fazer.

Apêndice A

XYZ

A.1 Apêndice 1

A fazer.

Referências Bibliográficas

[ACM03] D. Alur, J. Crupi, and D. Malks. *Core J2EE Patterns: Best Practices and Design Strategies*. Core Series. Prentice Hall PTR, 2003. 1

[AGC+16] MaurÃŋcio Aniche, Bavota G., Treude C., Van Deursen A., and Gerosa M. A validated set of smells in model-view-controller architectures. 2016. 2

[Anda] Activities. https://developer.android.com/guide/components/activities.html. Last accessed at 29/08/2016. 2

[Andb] Android: entenda as diferenÃğas entre o art e o dalvik. http://www.tecmundo.com.br/android/54387-android-entenda-diferencas-entre-art-o-dalvik.htm. Last accessed at 30/08/2016. 7

[Andc] Android studio. https://developer.android.com/studio/index.html. Last accessed at 30/08/2016. 7

[And14] Google android software spreading to cars, watches, tv. http://phys.org/news/2014-06-google-android-software-cars-tv.html, June 2014. Last accessed at 26/07/2016. 1

[And16] Ford terá apple carplay e android auto em todos os modelos nos eua. http://g1.globo.com/carros/noticia/2016/07/ford-tera-apple-carplay-e-android-auto-em-todos-os-modelos-nos-eua.html, 2016. Last accessed at 26/07/2016. 1

[App16a] App annie index market q1 2016: China takes japanâĂŹs #2 spot for ios revenue. https://www.appannie.com/insights/market-data/app-annie-index-market-q1-2016/, April 2016. Last accessed at 30/08/2016. 1, 7

[App16b] Number of available applications in the google play store from december 2009 to february 2016. http://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/, 2016. Last accessed at 24/07/2016. 1, 7

[Fow99] Martin Fowler. *Refactoring. Improving the Design of Existing Code.* Addison-Wesley, 1999. 4, 8

[Fow06] Martin Fowler. Code smell. http://martinfowler.com/bliki/CodeSmell.html, February 2006. 8

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software (Adobe Reader).* Pearson Education, 1994. 1

[Gro16] Worldwide smartphone growth forecast to slow to 3.1% in 2016 as focus shifts to device lifecycles, according to idc. http://www.idc.com/getdoc.jsp?containerId=prUS41425416, June 2016. Last accessed at 23/07/2016. 1

[Hec15] Geoffrey Hecht. An approach to detect android antipatterns. page 766âĂŞ768, 2015. 1, 4

[LVKM+] Mario Linares-VÃąsquez, Sam Klock, Collin Mcmillan, Aminata SabanÃľ, Denys Poshyvanyk, and Yann-GaÃńl GuÃľhÃľneuc. Domain matters: Bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps. 4

[MDA+] Umme Mannan, Danny Dig, Iftekhar Ahmed, Carlos Jensen, Rana Abdullah, and M Almurshed. Understanding code smells in android applications. 2, 4

[ML13] Roberto Minelli and Michele Lanza. Software analytics for mobile applications, insights & lessons learned. *In Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering*, 2013. 4

[Mob16] Gartner says worldwide smartphone sales grew 3.9 percent in first quarter of 2016. http://www.gartner.com/newsroom/id/3323017, May 2016. Last accessed at 23/07/2016. 1, 7

[Per16] Global mobile os market share in sales to end users from 1st quarter 2009 to 1st quarter 2016). http://www.statista.com/statistics/266136/

global-market-share-held-by-smartphone-operating-systems/, 2016. Last accessed at 24/07/2016. 1

[RB13] Jan Reimann and Martin Brylski. A tool-supported quality smell catalogue for android developers. 2013. 2, 5

[Ref99] *Refactoring: Improving the Design of Existing Code.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. 1

[Rie96] A.J. Riel. *Object-oriented Design Heuristics.* Addison-Wesley Publishing Company, 1996. 2

[RW12] N. Rozanski and E. Woods. *Software Systems Architecture: Working with Stakeholders Using Viewpoints and Perspectives.* Addison-Wesley, 2012. 1

[Sma16] Global smartphone sales to end users from 1st quarter 2009 to 1st quarter 2016, by operating system (in million units). http://www.statista.com/statistics/266219/global-smartphone-sales-since-1st-quarter-2009-by-operating-system/, 2016. Last accessed at 24/07/2016. 1

[Tsa10] Nikolaos Tsantalis. *Evaluation and Improvement of Software Architecture: Identification of Design Problems in Object-Oriented Systems and Resolution through Refactorings.* PhD thesis, University of Macedonia, August 2010. 8

[Ver13] Daniël Verloop. *Code Smells in the Mobile Applications Domain.* PhD thesis, TU Delft, Delft University of Technology, 2013. 2, 4, 5, 8

[Wika] Wikipedia android. https://pt.wikipedia.org/wiki/Android. Last accessed at 30/08/2016. 7

[Wikb] Wikipedia android runtime. https://en.wikipedia.org/wiki/Android_Runtime. Last accessed at 30/08/2016. 7