Diego Lins de Freitas

Uso de Padrões de Projetos no desenvolvimento de aplicativos Android

Faculdade Fucapi

Especialização em Engenharia de Software

Centro de Pós-Graduação e Extensão (CPGE)

Manaus-Am

2013, v0.6

Resumo

A plataforma android foi adotada por vários fornecedores de smartphones e tablets promovendo uma disseminação do sistema operacional muito abrangente. Formou-se um grande mercado a ser explorado com fornecimento de aplicativos para as mais variadas finalidades. Para atender essa demanda é necessário desenvolver aplicativos de qualidade e o mais rápido possível. Porém, por se tratar de pequenos aplicativos há pouca preocupação com a arquitetura. Os padrões difundidos na comunidade são focados na utilização de componentes e desenvolvimento da interação homem-máquina. Neste trabalho são apresentados alguns padrões de projetos aplicados ao desenvolvimento de aplicativos android dando um visão mais ampla de um aplicativo.

Palavras-chaves: Android. MVC. MVP.

Sumário

1 Introdução

A tendência do mercado de dispositivos móveis é aumentar. Segundo IDC, é esperado um crescimento de 32.7% na produção em 2013 (IDC, 2013b). O sistema operacional android é o líder dominando 75% desse mercado com mais de 162 milhões de smartphones produzidos e embarcados com android(IDC, 2013a). Baseado nesses dados é possível concluir que existe uma damanda no desenvolvimento de novos aplicativos que agregem valor à esses aparelhos.

Para produzir aplicativos de qualidade é necessário aplicar boas práticas de desenvolvimento de software. Levando-se em consideração que a linguagem de programação usada para o desenvolvimento na plataforma android é a Java, é natural que se aplique as práticas definidas pelos princípios de projeto orientado a objetos. Esses princípios guiam o desenvolvedor em como definir a estrutura interna do software afetando diretamente características de qualidade como manutenabilidade, performance e outras(YANG; TEMPERO; MELTON, 2008).

Os padrões de projeto são aplicados na engenharia de software como forma de reproduzir soluções para problemas recorrentes melhorando a manutenabilidade e o reuso de componentes de software(GAMMA et al., 1995).O uso de padrões de projetos é um meio de aplicar esses princípios.

Boas práticas de engenharia de software promevem o desenvolvimento dos artefatos que constituem o sistema de forma coesa. Os conjuntos desses artefatos podem ser classificados de acordo com suas responsabilidades, surgindo uma divisão clara em camadas. Um sistema pode ter diversas camadas responsáveis por acesso à um banco de dados, cominicação com serviços externos, encapsular regras de negócio. Uma dessas camadas que constituem um aplicativo androia é a camada de apresentação ou interface com o usuário.

Segundo Pressman (2006), “A interface com o usuário pode ser considerada o elemento mais importante de um sistema ou produto baseado em computador“. Tendo em vista isso, será tratado nesse trabalho os padrões para desenvolvimento da camada de apresentação de em aplicativos android.

1.1 Motivação

A motivação para a execução desta pesquisa sugiu da participação do autor em projetos para desenvolvimento de aplicativos android em uma instituição de pesquisa e desenvolvimento localizada em Manaus. O comprometimento com a qualidade promoveu

o uso de ferramentas de código aberto para monitoração da qualidade dos projetos, além do processo de testes adotado pela empresa. Com uma equipe composta por mais de 30 desenvolvedores produzindo aplicativos com diversas finalidades, houve a necessidades de padronização da arquitetura.

1.2 Problematização

Dado que existem uma vasta diversidade de padrões de projeto a serem aplicados no desenvolvimento de software, Qual padrão de projeto é aplicável para o desenvolvimento da camada de apresentação de aplicativos android para melhorar a qualidade do produto final?

1.3 Hipótese

O padrão de projeto Model-View-Presenter é aplicável no desenvolvimento da camada de apresentação de aplicativos android, gerando um aumento da qualidade do produto final.

1.4 Objetivos

Este trabalho fará um estudo sobre a arquitetura de aplicativos android com o propósito de melhorar a qualidade desses softwares, causando uma redução nos riscos relacionados as mudanças que acontecem durante o ciclo de desenvolvimento e promova a reutilização de componentes e que permita o incremento de funcionalidades com baixo impacto. Este estudo fornecerá insumos para que os desenvolvedores possam ter uma visão geral da aplicabilidade dos padrões de projetos e analisar quais técnicas contribuem para o seus projetos. Este trabalho tem como meta:

- Estudar os padrões de projeto que podem ser usados para a implementação da camada de apresentação de um aplicativo android.
- Avaliar os componentes do framework android, identificando seus papéis e responsabilidades de acordo com os padrões estudados, levando em consideração características que podem dificultar a aplicação dos padrões.
- Identificar os impactos nas características de qualidade do aplicativo de acordo com o paradigma da Orientação a objetos.
- Propor um referência de implementação para a camada de apresentação de um aplicativo android utilizando padrões de projetos.

1.5 Trabalhos Relacionados

Na dissertação de Türk (2009) é feita uma análise dos impactos na qualidade ao aplicar cinco padrões de projetos em um software de comunicação TCP/IP. Os resultados do trabalho citado mostram que a qualidade do software aumenta ao aplicar padrões de projetos sendo que o principal atributo de qualidade aferida é a manutenabilidade.

1.6 Contribuições

Tendo em vista que padrões de projetos descrevem soluções genéricas, este trabalho tem como principal contribuição uma interpretação do padrão de projeto Model-View-Presenter, proporcionando uma referência prática aplicada ao framework android.

A diversidade de projetos de software torna difícil inferir se as métricas podem ser consideradas um paramêtro para determinar a qualidade. O presente trabalho se propõe a avaliar a aderência do padrão MVP â projetos android através de indicadores de qualidade de código orientado a objetos.

1.7 Organização do Trabalho

O presente trabalho está estruturado em cinco capítulos dos quais este é o Capítulo 1 que apresentou a proposta do trabalho, contexto do problema, a motivação para estes estudo, os objetivos a serem alcançados e a estrutura do trabalho. O Capítulo 2 trata da metodologia aplicada neste trabalho mostrando o objeto de estudo a ser analisado e os passos a serem executados, enfatizando o método experimental. O Capítulo 3 discorre sobre a fundamentação teórica sobre padrões de projetos, métricas de qualidade orientada a objetos e as tecnologias com as quais o objeto de estudo foi desenvolvido. O Capítulo 4 contém uma análise do objeto de estudo e define como serão feitas as implementações e mostra os resultados da execução da pesquisa. O Capítulo 5 faz a análise final, conclusões do trabalho, e sugestões para trabalhos futuros.

2 Metodologia

Esta pesquisa se caracteriza como experimental pois é selecionado um objeto de estudo que será analisado do ponto de vista de métricas bem definidas. Essas métricas serão influenciadas pelos experimentos de refatoração embasados na revisão literatura existente sobre orientação a objetos e seus princípios.

Para validar a hipótese apresentada neste trabalho a pesquisa terá uma abordagem quantitativa através da análise de dados estatistícos de métricas que expresssam atributos de qualidade em software orientado a objetos. O conjunto de métricas a serem usadas nessa validação será o elaborado por Chidamber e Kemerer (1994).

Essas medidas serão coletadas através de um procedimento experimental em laboratório utilizando um processo iterativo-incremental para executar refatorações no código do objeto de estudo afim de introduzir o padrão de projeto Model-View-Presenter e a cada iteração será feita a coleta das métricas. Para executar esse processo será identificado um conjunto de funcionalidades que o aplicativo atende, relacionda com a camada de apresentação com a qual o usuário interage.

2.1 Objeto de Estudo

O projeto a ser refatorado será o aplicativo de Contatos do android[1], que é um dos aplicativos básicos pré-instalados com o sistema operacional. Este projeto é opensource mantido pelo The Android Open Source Projeto suportado pela Google com contribuições de desenvolvedores do mundo todo.

Os critérios para a escolha do objeto de estudo são:

1. Tamanho/Complexidade - Um projeto muito complexo iria inviabilizar a pesquisa devido ao esforço para fazer a refatoração. Métricas coletadas apartir de projetos simples e triviais não forneceriam dados suficientes para uma análise satisfatória.

2. Código Aberto - Além de permitir o acesso ao código fonte sem limitações para a pesquisa, tem a contribuição de vários desenvolvedores com experiência e formação em programação diversificada que se refletem no código fonte.

3. Origem do projeto - A escolha de um aplicativo mantido sobre o mesmo gerenciamento que o sistema operacional android foi feita com o intuito de fazer a pesquisa em um código fonte que expressasse as técnicas e práticas de programação difundidas nesse ecossistema.

[1] https://android.googlesource.com/platform/packages/apps/Contacts

2.2 Processo de Experimentos

A Figura 1 demonstra as atividades do processo de experimentação adotado:

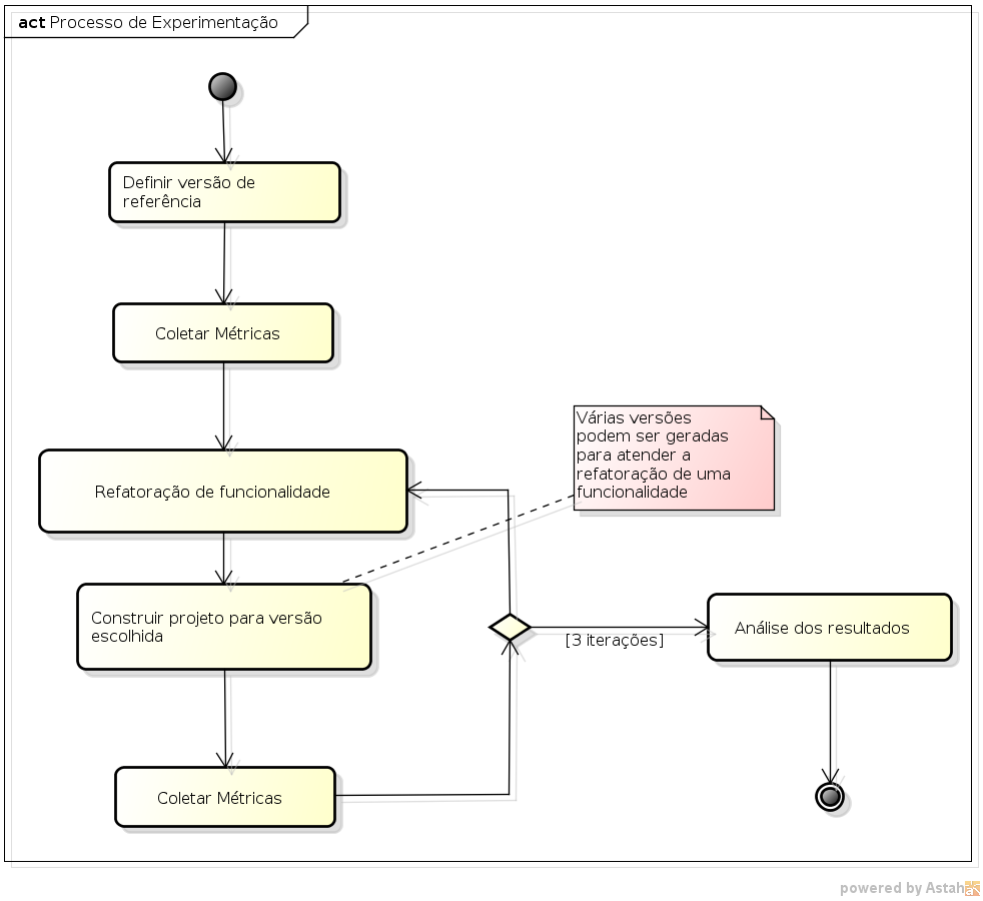

Figura 1 – Processo de Experimentação Fonte: Próprio Autor

Definir versão de referência - Delimitar um marco do estado do código no repositório.

Refatoração de funcionalidade Esta atividade tem como objetivo aplicar os padrões propostos em uma funcionalidade do aplicativo.

Construir proejto Excutar o processo de construção do projeto que inclui a compilação das classes para fazer a coleta das métricas.

Coleta de Métricas Este passo tem como objetivo fazer a coleta das métricas do código que se encontra no repositório a partir de uma revisão para fazer a avaliação dos efeitos da refatoração na qualidade do código.

Análise dos resultados Discussão dos impactos das alterações executadas nas métricas.

3 Referencial Teórico

3.1 Princípios e Padrões de Projetos

Desenvolver software orientado a objetos é um desafio. Criar uma representação computacional de uma faceta da realidade em que seus constituintes trabalhem de forma harmoniosa para atingir as necessidades que o software se propõe a atender requer experiência, conhecimento do domínio do problema e um processo de análise e projeto. Apesar de existir várias abordagens para se conceber um sistema orientado a objetos(EVANS, 2004),(GOMAA, 2011) um sistema bem contruído apreseta características fundamentais como alta coesão e baixo acoplamento

designReusableClasses

Um padrão, dentro do contexto de estudo deste trabalho pode ser definido como uma técnica efetiva cuja a sua aplicabilidade é aceita e difundiida dentro de uma área de conhecimento com a intenção de atingir um objetivo(METSKER; WAKE, 2006).

Em desenvolvimento de software, o catálogo mais difundido de padrões de projetos orientado a objetos é elaborado por Gamma et al. (1995), contendo um total de 23 padrões formalmente documentados que acumulam experiências bem sucedidas em diversos sistemas. Esses padrões têm a seguinte classificação:

Criacionais Padrões que definem como criar novas instâncias de classes.

Estruturais Foca na estruturação das classes e objetos.

Comportamentais Definem como as classes e objetos interagem entre si e suas responsabilidades.

Analisando a forma como um padrão de projeto é concebido, com uma definição dos papéis de cada elemento participante e como eles interagem entre si, pode-se concluir que o uso dos padrões de projeto promove maior coesão, melhor separação de interesses e baixo acoplamento no sistema. Todas essas características são muito importantes e contribuem para um software de melhor qualidade.

3.2 Métricas de qualidade OO

Com o advento de novas técnicas de desenvolvimento de software é necessário obter informações do impacto dessas inovações nos resultados de um projeto. Com esse

objetivo Chidamber e Kemerer (1994) elaboram um conjunto de métricas para mensurar a qualidade de sistemas desenvolvido usando o paradigma orientado a objetos que não se limitasse a uma linguagem de programação, fácil de coletar e com forte embasamento teórico na ontologia de bunge[1]. This model is used to analyze some static and dynamic properties of an information system and to examine the question of what constitutes a good decomposition of an information system(WAND; WEBER, 1990)As métricas são:

Acoplamento entre objetos (CBO) Número de classes que ela depende por meio da relação de composição. Uma classe está acoplada a outra quando o método de um classe invoca o método de uma variável de instância de outra classe que gera uma dependência entre essas classes. Quanto maior essa dependência, mais difícil é reutilizar esses componetes em outras partes do sistema, além do aumento do risco de efeitos colaterais ocorrerem ao modificar uma classe altamente acoplada.

Ausência de coesão dos métodos(LCOM) Usado para avaliar a coesão de uma classe através da similaridade entre seus métodos. Um método tem similaridade com outro quando a intercessão entre os conjuntos de atributos usados por ambos os métodos tem cardinalidade maior que zero. LCOM mostra esses conjuntos nulos indicando os métodos que usam esses atributos deveriam ser implementados em outra classe. Essa similaridade expressa a coesão da classe.

Profundidade na árvore de herança (DIT) Nível de uma classe na hierárquia de herança. Reflete o número máximo de elementos pai dentro da aŕvove de classes até a raiz, o que aumenta a complexidade conforme a quantidade de elementos envolvidos se eleva, diminuindo a previsibilidade do comportamento da classe com vários métodos e atributos sendo herdados, principalmente com o uso de sobrecarga de métodos.

Métodos por Classe (WMC) Serve para expressar o nível de complexidade de uma classe basseado no número de métodos que ela possui. Isso afeta o esforço de manutenção da classe, além de impactar nas classe filhas que herdarão esses métodos e também é um indicativo de que a classe tem métodos específicos dificultando o seu reuso.

Número de classes filhas (NOC) Número de subclasses imediatas de uma classe. Essa medida é um indicativo de mau uso de herança conforme seu valor aumenta e mostra o impacto que uma classe pode ter no sistema requerendo maior atenção e testes.

Response sets for Class (RFC) Quantidade de métodos que são executados quando um objeto recebe uma mensagem, incluindo os métodos de outras classes.

1

Todas essas métricas tem uma relação inerente com a coesão e acoplamento dos objetos, sendo uma forma confiável para a análise da qualidade em sistemas orientados a objetos. Os aplicativos desenvolvidos para a plataforma android são escritos usando a linguagem de programação Java que emprega esse paradigma de desenvolvimento, o que justifica o uso das métricas de Chidamber e Kemerer (1994) para validação dos projetos orientados a objetos.

3.3 Model View Controller

O padrão Model View Controller surgiu como uma solução genérica para que usuários de uma sistema de planejamento manipulem dados complexos Reenskaug (1979). Posteriormente, Krasner e Pope (1988) implementam um framework MVC para o ambiente gráfico da linguagem de programação Smalltalk-80 como uma forma de promover a reusabilidade e plugabilidade.

Segundo Reenskaug (1979) o principal objetivo do MVC "...é representar o modelo mental do usuário de um espaço de informações relevantes e permitir que o usuário inspecione e altere esta informação."(tradução livre). Esse modelo mental é como o usuário percebe o dominio do problema que está inserido no qual executará suas atividades sobre dados de seu interesse. Para que o usuário de um sistema de informação possa interagir com a represetação computacional de seu modelo mental três componetes são definidos:

Models - É o compoente constituído de uma composição de classes que implementam as regras de negócio referentes as funcionalidades que o programa provê, representa o conhecimento que o usuário tem e como manipula-lo. Atende mensagens da view requisitando seu estado e mensagens do controller para mudar seu estado,

Views - Representação específica de um model na interface com o usuário, é responsável por todo a manipulação visual, recuperando um estado do model e exibindo os dados, podendo ser composta por sub-views e ser parte de views mais complexas.

Controllers - Interpreta a as ações do usuário provenientes de um dispositivo de entrada(Teclado, Mouse) alterando estado da view ou do model.

Krasner e Pope (1988) descrevem a estrutura do MVC onde a view tem seu controller exclusivo mantendo uma dependência cíclica entre ambos. Tanto a View quanto o Controller tem referências diretas para o model por meio de atributos de classe, porém, o model não deve conhecer seus respectivos pares de View-Controller para promover maior reuso de código e encapsulamento do model. As alterações do estado do model são feitas na maioria das vezes pelo controller, e o model é responsável por notificar todas as views que o representa para que se atualizem refletindo o novo estado. No caso de um model ser usado por vários pares de View-Controller as mensagens de notificação de um novo

estado do model podem ser parametrizadas assim cada view pode verificar se a alteração é de seu interesse.

Segundo Fowler (2002) "...esta separação da apresentação e modelo é uma das mais fundamentais heristicas de bom projeto de software"(tradução livre). O controller poderia ser o responsável por publicar as alterações no estado do model devido sua relação direta com o mesmo, mas em casos onde o model é alterado por outro componente que não é um dos controladores que os utilizam, é necessário que o model conheça as views que devem ser notificados do novo estado. Para que essas alterações de estado sejam propagadas a view e o controller são registrados como dependentes de seu model. O padrão é descrito dentro do contexto no qual o surgiu levando em consideração caracteristicas espefíficas da linguagem de programação que dão suporte à implementação dos três componentes como por exemplo o gerenciamento dos objetos que são dependentes do model definido na classe Objet que o model deve extender. A Figura 2 esclarece a interação entre os componetes.

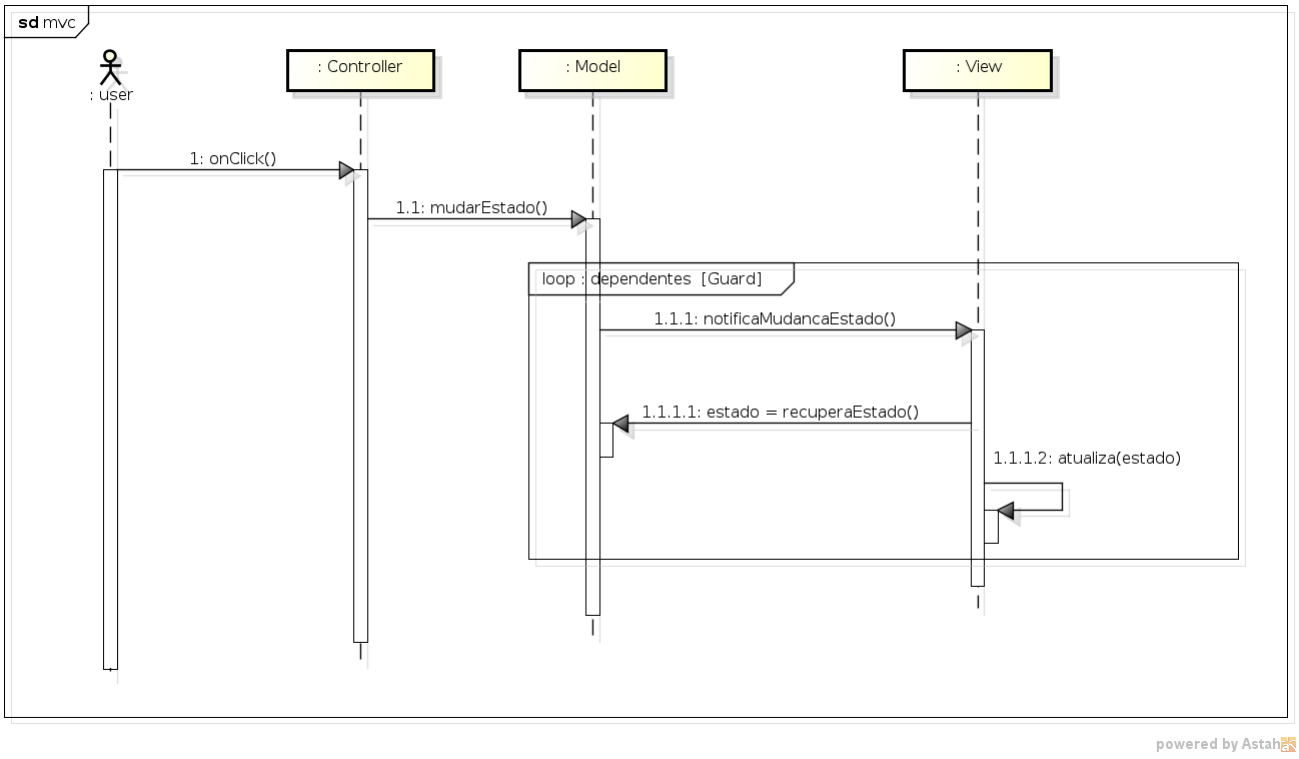

Figura 2 – Diagrama de Sequência do MVC/Fonte: Próprio Autor

Gamma et al. (1995) cita Krasner e Pope (1988) fazendo uma análise dos objetos que compõem o MVC relacionando-os com outros padrões de proejto descritos em seu catálogo. O desacoplamento entre a View e o Model, somado à propagação das mudanças de estado no model para os objetos regitrados como dependentes do model pode ser descrito como uma implementação do padrão Observer. O padrão Observer define uma estrutura em que um componente que precisa publicar mudanças em seu estado, detém

a referência para uma lista de objetos a serem notificados. A hierarquia de views é um exemplos de Composite pois uma view pode ser constituída por sub-views para compor views complexas. No Composite um conjunto de componentes podem ser tratadas de forma encapsulada onde cada implementa as mesmas abstração. O padrão Strategy define uma abstração cuja as implementações podem ser trocadas de acordo com algum critério, esse conceito pode ser aplicado ao controller que encapsula o algoritmo que vai alterar a View e o Model, permitindo sua substituíção por uma outra implementação que deixa de responder às interações com o usuário.

Segundo Krasner e Pope (1988) o Model "...pode ser simples como um valor numério inteiro (como o modelo de um contador) ou um valor literal (como o modelo de um editor de texto), ou pode ser um objeto complexo"(tradução livre). O model pode ser implementado usando o pardrão Facade para simplificar as interações com o Model dependendo da complexidade do domínio que ele representa.

3.4 Model View Presenter

O MVP é um modelo de programação para implementação de interfaces com o usuário desenvolvido como um framework para C++ e Java, criado por uma subsidiária da IBM chamda Taligent,Inc. Este padrão é baseado no MVC e descreve vários componentes que tem as responsabilidades de como gerenciar os dados da aplicação e como o usuário interage com esses dados, tendo como objetivo promover o encapsulamento do Model , reuso de lógica de negócio e o polimorfismo da View.

Model Tem as mesmas responsabilidades que o Model definido pelo MVC.

Selections - Abstração para selecionar um subconjunto dos dados existentes no model.

Commands Representa as operações a serem executadas sobre uma Selection do Model.

View Responsável por exibir o model assim como no MVC.

Interactor Mapeia os interações do usuário na view como eventos do mouse.

Presenter O papel do presenter é interpretar o eventos iniciados pelo usuário executando a lógica de negócio correspondente implementada em um command para manipular o model (POTEL, 1996).

Os conceitos do MVP são descritos em Potel (1996) de forma genérica permintindo interpretações para uma implementação efetiva. Bower e McGlashan (2000) descreve a implementação de um framework para Dolphin Smalltalk[2] adotando os conceitos do MVP

[2] Implementação da Linguagem de programação Smalltalk - http://www.object-arts.com

onde salienta que a maioria dos sistemas operacionais com ambiente gráfico fornece um conjunto de componentes (Widgets) no qual está contido a responsabilidade do controller.A maior parte do comportamento do caso com o usuário é implementada no Presenter que está diretamente associado à View.

Ainda acerca das responsabilidades do Presenter, Fowler (2006) descreve o que é chamado de Passive View, onde toda a lógica do comportamento da view é implementado no presenter deixando a view enxuta com o intuito de isolar ao máximo a API gráfica do resto da aplicação. Dessa forma o model não se comunica com a view por meio do observer pattern, sendo que a view séra atualizada pelo presenter como pode ser observado na Figura 3.

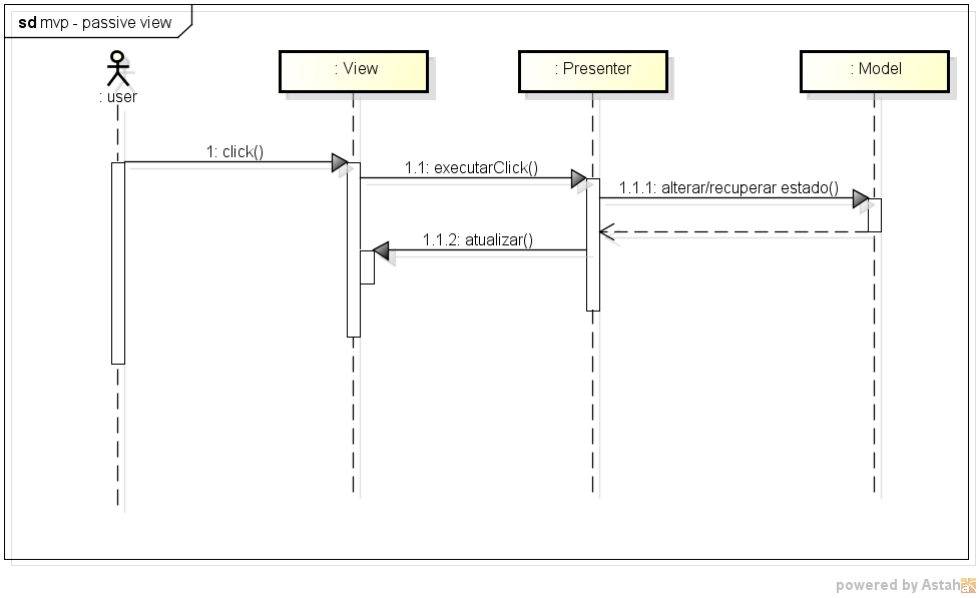

Figura 3 – Passive View/Fonte: Próprio autor

MVP se adequa melhor as apis gráficas existentes e define de forma mais clara os componetes necessários para desenvolver uma aplicação, sendo o ponto de maior discussão reside em quais os limites das responsabilidades no que tange a mediação do Model e a View por parte do Presenter.

3.5 Framework Android

O android é um sistema operacional baseado no linux mantido pela Google para ser embarcado em dispositivos podendo ser aplicado em carros, televisão, placas controladoras mas seu destaque é a utilização em smartphones e tablets, que é o foco deste trabalho. A plataforma é contituída por API's e frameworks tendo em sua base o sistema operacional e seus drivers seguido da máquina virtual que executa os aplicativos android e bibliotecas auxiliares e aplicativos básicos como é demonstrado na figura 4.

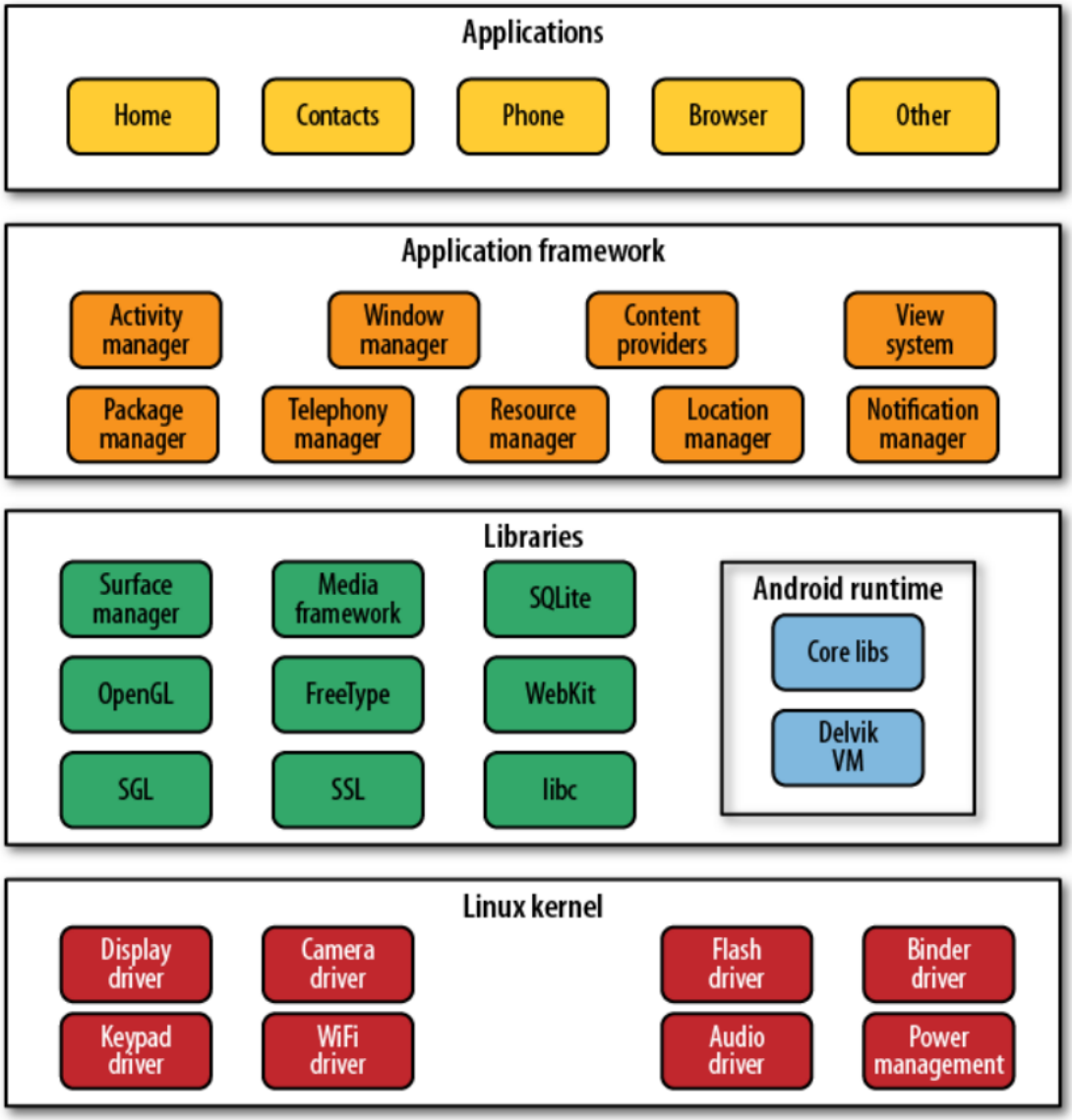

Figura 4 – Android Stack/Fonte: Learning Android

Para desenvolvimento é usado a api disponível no sdk que define os blocos de construção de um aplicativo, a saber:

Activity Representa uma atividade que o usuário executa no aplicativo em um determinado momento. É um agregador de componetes visuais e responde à interações do usuário.

Fragment Representa uma parte de interface com o usuário em uma Activity.

Service Responsável por executar uma operação sem interface gráfica indicado para processamentos longos como por exemplo a execução de uma música ou download de de arquivos.

Broadcast Receiver Implementação do padrão publish/subscribe

Content Provider Usado para expor dados de uma aplicativo para outros aplicativos. Os dados podem ser provenientes de qualquer forma de armazenamento como um arquivo ou banco de dados.

ApplicationContext Representa a aplicação em execução provendo acesso a recursos.

AsyncTask Usado para implementar computação paralela evitando o uso da linha de execução principal do aplicativo que é respoonsável por tratar a interações com o usuário.

Com base nos componentes de framework e literatura revisada é possível fazer uma análise dos mesmos e projetar uma camada de apresentação utilizando o padrão MVP para ser usada como referência de implementação a ser aplicada.

4 Execução da Pesquisa

4.1 Ferramentas usadas

O código será versionado no Github[1] onde será feito o gerenciamento das versões de cada iteração. As ferramentas utilizadas para a refatoração serão a IDE Eclipse(Juno) com plugin ADT v21 para facilitar a edição do código e ferramentas de construção do projeto existentes no próprio repositório do android tendo em vista que todo o processo de compilação e empacontamento não visa ser usado em uma ide. Para fazer a coleta das métricas é necessario que a ferramenta analise código java e contemple todas as métricas descritas na seção 3.2. O programa Chidamber and Kemerer Java Metrics[2] atende esses critérios, além de ser um projeto de código-aberto.

4.2 Análise do objeto de estudo

O aplicativo a ser refatorado tem funcionalidades para gerenciamento de contatos. Dentro deste conjunto de casos de uso será feratorado o pacote referente ao gerenciamento de grupos de contatos presente no pacote **com.android.contacts.group** que contém componentes de tela para interação com o usuário a saber:

GroupDetailFragment.java Exibe os dados de um grupo de contatos.

GroupBrowseListFragment.java Fonece uma lista de grupos.

GroupEditorFragment.java Disponibiliza um formulário para edição dos dados de um grupo.

Estas intefaces são usadas dentro de activities que controlam uma parte do fluxo de interação e se comportam de forma diferente conforme o tipo de dispositivo móvel utilizado. Devido a essa complexidade, não será feita nenhuma alteração na interface pública dos componentes refatorados evitando efeitos colaterais em outras partes do aplicativo.

Os componetes elencados contém código não somente relacionado com a lógica de apresentação como também interagem diretamente com classes destinadas ao acesso de dados e serviços existentes nas dependências do projeto, por exemplo, gerenciamento de contas do usuário.

[1] https://github.com/diegofreitas/platform_packages_apps_contacts
[2] https://github.com/dspinellis/ckjm

Cada iteração compreenderá na na refatoração de cada um dos componentes descritos. O marco de referência de dados das métricas presentes na tabela 1 será feita a partir da versão `4.4.2_r1` do aplicativo.

Métrica	Média
WMC	8.5161290323
DIT	0.7741935484
NOC	0
CBO	10.1612903226
RFC	23.7419354839
LCOM	57.4838709677

Tabela 1 – Métricas CK para linha de base

4.3 Arquitetura Proposta

Será aplicado nos experimentos a variação do padrão MVP chamada Passive View, pois dessa forma, o Model não precisa publicar alterações de seu estado para a view, dessa forma evita-se alterações no código referente às classes que fazem parte da camada de Model do aplicativo de contatos. A organização do código fonte no repositório dificulta a implementação porque esses componentes estão localizados fora do projeto afetado e são compartilhados.

As classes que extendem Fragment terão a responsabilidade da View pois é neste componente que a interface com o usuário é construída. A classe Activity fornece vários métodos para recuperação de recursos de imagens, textos, inicialização de serviços, entre outros. Isso ocorre porque a classe Activity é uma subclasse de Context, herdando diversos métodos não relacionados ao gerenciamento da interface.

Segundo Reenskaug (1979) "... Os papéis da View e o Controller podem ser exercidos pelo mesmo objeto quando eles estão muito acoplados. Exemplo: A Menu.", porém isso requer um boa análise do problema em questão para decidir o nível de granularidade que esses componentes podem ter. Portanto, é recomendável manter sempre essa separação para manter uma boa coesão nas classes. O Presenter será uma classe auxiliar à view e pode ser implementada como uma classe java simples.

4.4 Experimentos

4.4.1 WMC

Como fazer referência à tabela e ao gráfico?

Iteração	Média
Baseline	8.5161290323
Iteração 1	8.875
Iteração 2	9
Iteração 3	8.9393939394

Tabela 2 – Valores da métrica

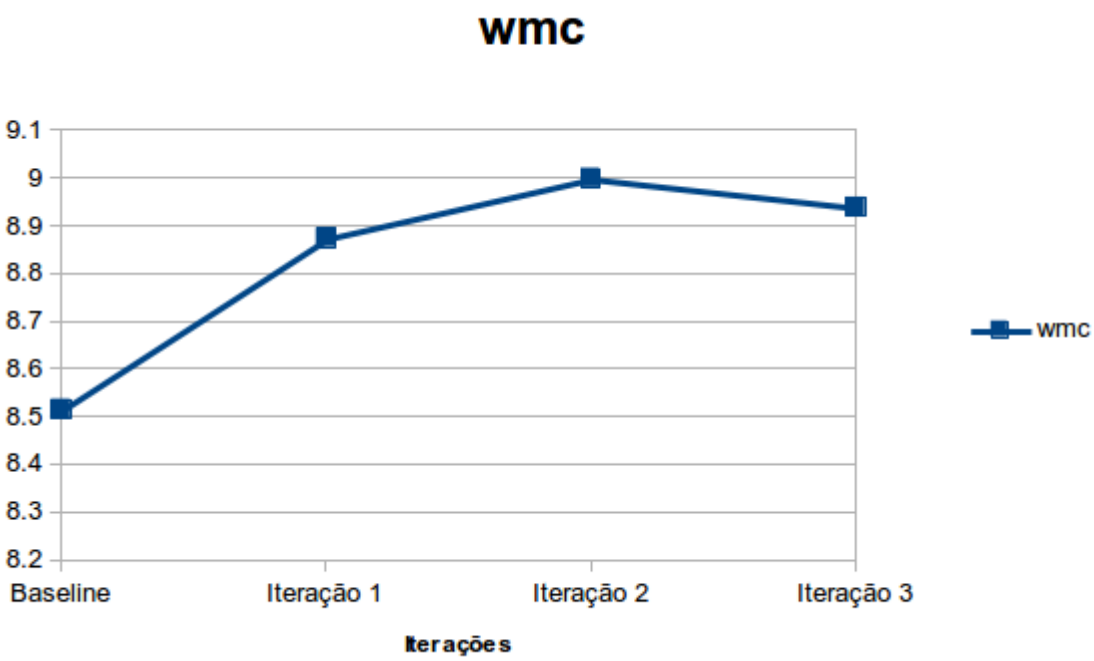

Figura 5 – Gráfico da métrica WMC/Fonte: Próprio autor

Segundo Chidamber e Kemerer (1994) ". . . The larger the number of methods in a class the greater the potential impact on children, since children will inherit all the methods defined in the class. . . . Classes with large numbers of methods are likely to be more application specific, limiting the possibility of reuse." as classes analisadas são de interface, portanto implementão uma interação com o usuário bem específica que não serão reutilizadas por meio de herança.

4.4.2 DIT

A figura 6 mostrar um aumento no valor da métrica DIT, que deve ser mantido baixo pois quanto mais abaixo na hierarquia de heraça a classe estiver, menos previsível será seu comportamento devido a quantidade de classes acima, entretanto o aumento no DIT se deve ao fato que o ckjm considera a implementação de interface como herança. A interface define somente o contrato que a classe de implementar não havendo nenhuma implementação que pudesse interferir no comportamento da classe, portanto essa alteração

no DIT não deve ser considerada.

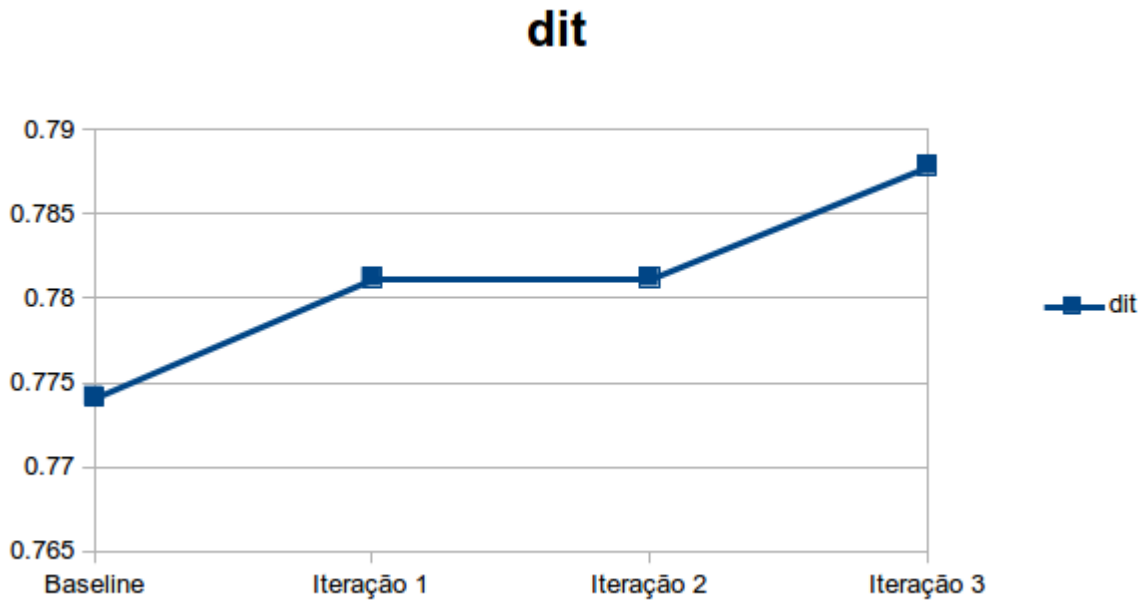

Figura 6 – Valores de DIT/Fonte: Próprio autor

4.4.3 NOC

Nenhuma classe foi herdada para a aplicação do padrão, portanto, essa métrica permaneceu intacta durante as iterações.

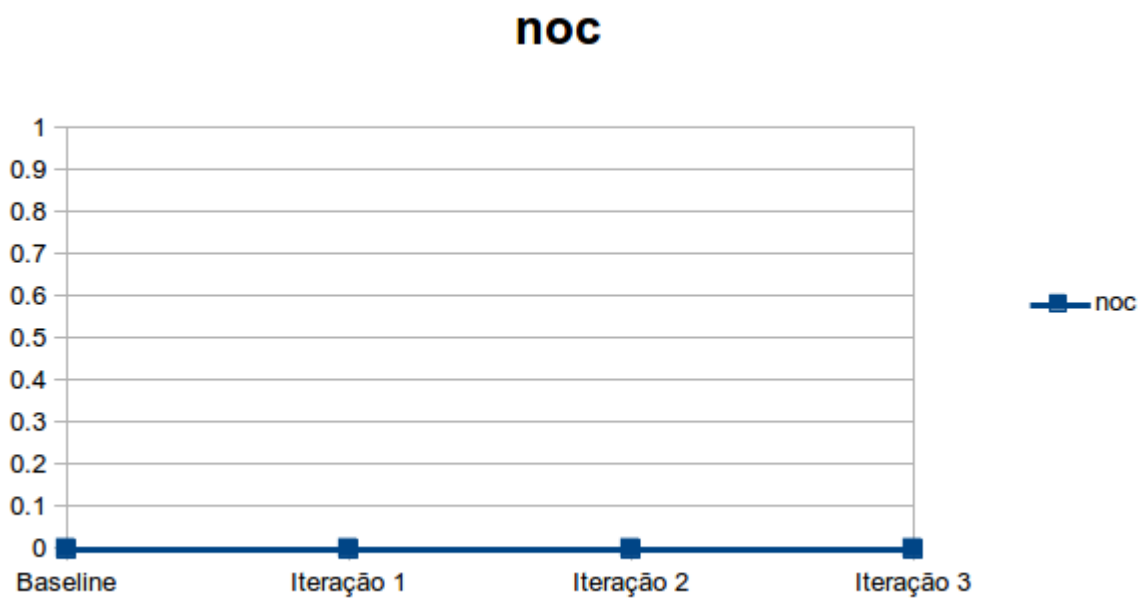

Figura 7 – Valores de NOC/Fonte: Próprio autor

4.4.4 CBO

Na primeira iteração foi aplicado o padrão em um componente mais simples e foi possível remover qualquer dependência que não fosse relacionada a interface, entretanto, a maior queda ocorreu em ficando a cargo do da experiência do desenvolvedor identificar estes cenários específicos. Nas iterações seguintes as interfaces alteradas são mais complexos,

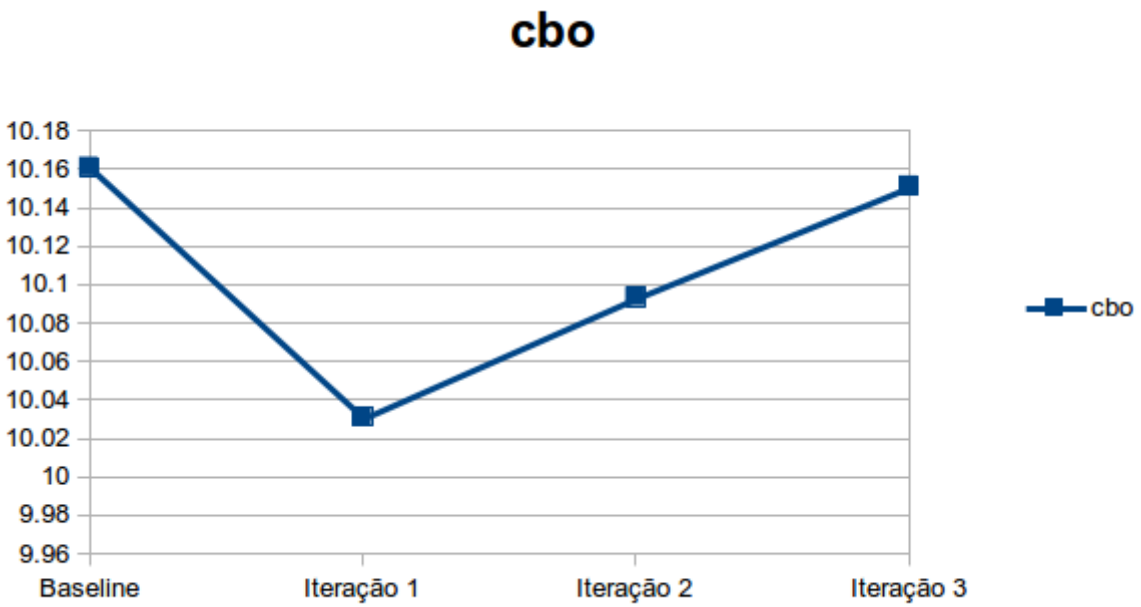

Figura 8 – Valores de CBO/Fonte: Próprio autor

4.4.5 RFC

A métrica RFC tende a aumentar a cada iteração o que sugere um aumento da complexidade do código conforme é apresentado na Figura 9.

4.4.6 LCOM

A Figura 10 mostra uma queda siginificativa na métrica LCOM. Isto indica que a coesão do código melhorou após cada iteração.

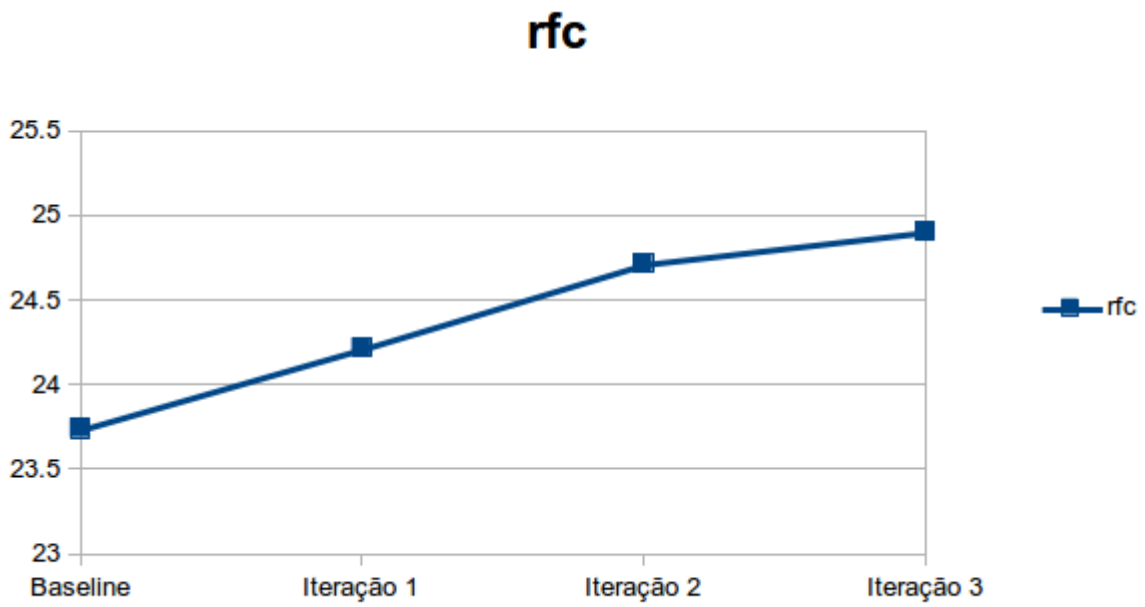

Figura 9 – Valores de RFC/Fonte: Próprio autor

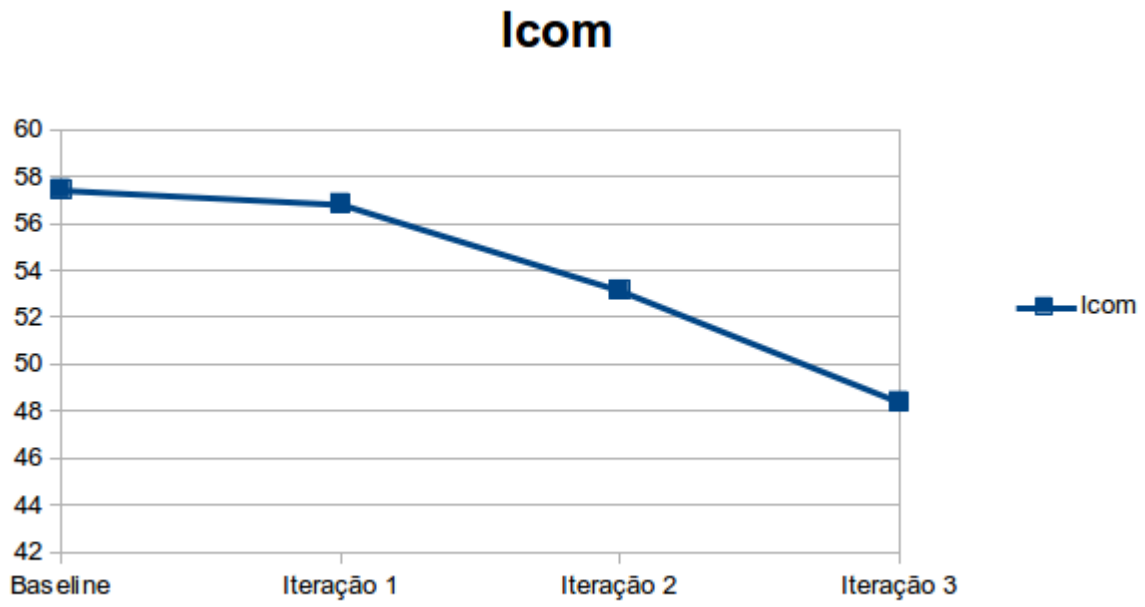

Figura 10 – Valores de lcom/Fonte: Próprio autor

5 Conclusão

5.1 Discussão dos Resultados

As classes destinadas à implementação da interface no framework android fornecem acceso a recursos que servem para implementação de responsabilidades não relacionadas com a interface.

Essa característica do framework android leva à implementação da camada de View com diversas responsabilidades que não são inerentes à interação com o usuário. Isso dificultou a refatoração pois as classes que exercem o papel de presenter necessitam interagir com as classes de view para acessar esses recursos, além de atualizar o estado da view. O uso do padrão de injeção de dependência[1] pode ser aplicado para acessar esses recursos e serviços sem a necessidade de interação com a classe de Veiw.

Ao seguir o padrão conforme descrito por Fowler (2006), a granularidade dos métodos aumenta pois cada um dos componentes implementão uma parte do caso de uso aumentando o números de métodos que se reflete na métrica WMC, isso afeta também a métrica RFC pois esses métodos estão relacionados entre si aumentando a troca de mensagens entre a view e o presenter. Um ponto positivo sobre isso é o fato de que métodos complexos com muitas linhas de código foram desmembrados em métodos menores e mais simples implementados tanto na view como no presenter.

Houve diminuição na métrica CBO nas classes alteradas pois diversas responsabilidades que utilizam essas dependências foram movidas para a classe de presenter. Analisando de forma geral, essas dependências permanecem no pacote além de ser criada mais uma acoplamento entre a view e a nova classe presenter. Os experimentos demonstraram que a aplicação do padrão MVP promoveu de forma significativa maior coesão no aplicativo.

O objeto de estudo deste trabalho não foi concebido com a aplicação intensiva dos conceitos da forma como foram apresentados. Os resultados das métricas

5.2 Trabalhos Futuros

Existem outros padrões de projetos para o desenvolvimento da camada de apresentação de um software que não foram analisados nesse trabalho a saber: MVVM, MVP-VM, MVPC. Essas variações no padrão MVC surgiram em contextos diversificados e podem agregar algum benefício à qualidade do aplicativo. Este trabalho não aborda o impacto

[1] http://en.wikipedia.org/wiki/Dependency_injection

do padrão mvp em outras métricas de qualidade de código. Não foi feita uma avaliação dos impactos na performance do aplicativo devido ao uso do padrão MVP. A inclusão de mais objetos interagindo trocando mensagens pode depreciar a performance levando-se em consideração sua execução em ambientes mais restritos como um aparelho móvel.

Referências

BOWER, A.; MCGLASHAN, B. Twisting the triad: The evolution of the dolphin smalltalk mvp application framework. In: *ESUG2000 Conference*. Southampton, UK: [s.n.], 2000. Disponível em: <http://www.objectarts.com/>. Citado na página 15.

CHIDAMBER, S.; KEMERER, C. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, v. 20, n. 6, p. 476 – 493, 6 1994. Citado 4 vezes nas páginas 9, 12, 13 e 21.

EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. [S.l.]: Addison-Wesley, 2004. Citado na página 11.

FOWLER, M. *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley, 2002. ISBN 0321127420. Citado na página 14.

FOWLER, M. *GUI Architectures*. 2006. GUI Architectures. Disponível em: <http://www.martinfowler.com/eaaDev/uiArchs.html>. Acesso em: 20.8.2013. Citado 2 vezes nas páginas 16 e 25.

GAMMA, E. et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. [S.l.]: Addison Wesley, Boston, 1995. (Professional Computing Series). Citado 3 vezes nas páginas 5, 11 e 14.

GOMAA, H. *Software modeling and design : UML, use cases, patterns, and software architectures*. [S.l.]: Addison-Wesley, 2011. Citado na página 11.

IDC. *Android and iOS Combine for 92.3System Shipments in the First Quarter While Windows Phone Leapfrogs BlackBerry*. 2013. IDC - Worldwide Quarterly Mobile Phone Forecast. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS24143513>. Acesso em: 9.7.2013. Citado na página 5.

IDC. *Smartphones Expected to Grow 32.7% in 2013 Fueled By Declining Prices and Strong Emerging Market Demand*. 2013. IDC - Worldwide Quarterly Mobile Phone Forecast. Disponível em: <http://www.idc.com/getdoc.jsp?containerId=prUS24143513>. Acesso em: 9.7.2013. Citado na página 5.

KRASNER, G.; POPE, S. A description of the Model-View-Controller user interface paradigm in the smalltalk-80 system. *Journal of Object Oriented Programming*, n. 3, p. 26–49, 1988. Citado 3 vezes nas páginas 13, 14 e 15.

METSKER, S. J.; WAKE, W. C. *Design Patterns in Java*. 2. ed. Upper Saddle River, NJ: Addison-Wesley, 2006. ISBN 978-0-321-33302-5. Citado na página 11.

POTEL, M. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Javaquot;, Taligent Inc.* [S.l.], 1996. Citado na página 15.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. [S.l.]: Mcgraw-hill Interamericana, 2006. Citado na página 5.

REENSKAUG, T. M. H. *Thing-Model-View-Editor – an Example from a planningsystem*. [S.l.], 1979. Acesso em: 1.8.2013. Citado 2 vezes nas páginas 13 e 20.

TüRK, T. *THE EFFECT OF SOFTWARE DESIGN PATTERNS ON OBJECT-ORIENTED SOFTWARE QUALITY AND MAINTAINABILITY*. Dissertação (Mestrado) — Middle East Technical University: METU, Çankaya Ankara/ Turquia, Setembro 2009. Citado na página 7.

WAND, Y.; WEBER, R. An ontological model of an information system. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 16, n. 11, p. 1282–1292, nov. 1990. ISSN 0098-5589. Disponível em: <http://dx.doi.org/10.1109/32.60316>. Citado na página 12.

YANG, H. Y.; TEMPERO, E.; MELTON, H. An empirical study into use of dependency injection in java. In: *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*. [S.l.: s.n.], 2008. p. 239–247. ISSN 1530-0803. Citado na página 5.