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 tablet 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 mas por se tratar de pequenos aplicativos há pouca preocupação com a arquitetura é deixada de lado. Os padrões difundidos na comunidade são focados na utilização de componentes e desenvolvimento da interface. Neste artigo é apresentado 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 handset é aumentar segundo IDC é experado um crescimento de 32.7% na produção(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). Baseados nesses dados podemos 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 em consideração que a linguagem de programação usado 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 as 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) então o uso de padrões de projetos é um meio de aplicar esses princípios. 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 proejtos a serem aplicados no desenvolvimento de software, Qual padrão de projeto é aplicável para o desenvolvi-

mento 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 reduzindo riscos relacionados as mudanças que acontecem durente 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 seu trabalho. 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 padões 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.
- Elaborar 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.

No presente trabalho será apresentado como objeto de estudo um projeto de open-source toda a pesquisa é reproduzível.

1.6 Contribuições

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

A diversidade de projetos de software torna difícil inferir se as métricas podem ser consideradas um paramêtro para determinar a qualidade. Para validar a aplicabilidade do padrão MVP métricas orientadas a objetos foram usadas dentro do contexto do framework android, contribuindo com mais uma estudo dessas métricas acrescentando algo para a ciência. rsrs

1.7 Organização do Trabalho

Será descrito a metodologia, definicão do projeto, experimentos, conclusões.

2 Metodologia

Esta pesquisa se caracteríza como aplicada pois tem fins práticos ao realizar uma revisão na literatura existente sobre orientação a objetos e seus princípios bem como referências sobre padrões de projetos para proporcionar o embasamento que auxiliará na identificação dos componetes do android que podem assumir as responsabilidades definidas nos padrões e definir como implementa-los.

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 coleata das métricas. Para executar esse processo será identificado um conjunto de funcionalidades que o aplicativo atende, relaciondas 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.(qual é a licença do projeto).

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. (o que foi usado para medir a complexidade. e qual o tamanho do objeto de estudo).

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 "guarda-chuva" que o sistema operacional android foi feita com o intuito de fazer a pesquisa

[1] https://github.com/android/platform$_{p}ackages_{a}pps_{c}ontacts$

em um código fonte que expressasse as técnicas e práticas de programação difundidas nesse ecosistema.

2.2 Processo de Experimentos

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

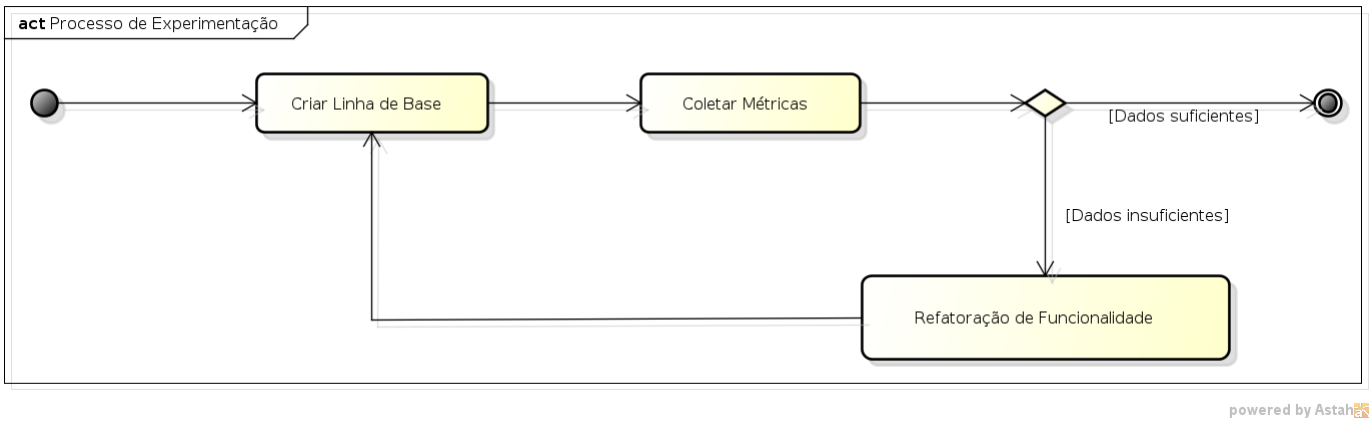

Figura 1 – Processo de Experimentação Fonte: Próprio Autor

Criação uma linha de base - Delimitar um marco do estado do código no repositório.

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

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

Esta pesquisa se propõe a executar três iterações.

2.3 Ferramentas

Para elaborar uma documentação da implementação de referências usando será usada a notação UML v2 com o uso da ferramento Astah. O código será versionado no Github onde será feito o gerenciamento das linhas de base. As ferramentas a serem utilizadas para a refatoração, IDE Eclipse(Juno) com plugin ADT v21. As métricas serão coletadas com o programa ckjm[2].

**** Salientar que tudo que for feito neste trabalho é reproduzivel estará tudo disponível no github.

[2] http://www.spinellis.gr/sw/ckjm/

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 horminiosa 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. Apesear de existir várias abordagens para se conceber um sistema orientado a objetos(EVANS, 2004),(GOMAA, 2011) um sistema bem contruído apresetam características fundamentais. Coesão, acoplamento etc

designReusableClasses

Um padrão dentro do contexto de estudo deste presente trablho 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 densenvolvimento 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 tem 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 definindo papéis de cada elemento participante e como eles interagem entre se podemos 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 contribuem para um software de melhor qualidade

3.2 Qualidade de Software

Seguir definição da ISO/IEC 9126-1 Funcionalidade. Esta pesquisa assume que o aplicativo está funcional Confiabilidade. exitem ferramentas que coletam informações

importantes como validação de variáveis nulas e tratamento de exceção mas estão fora do escopo OO. Usabilidade. A usabilidade do aplicativo não será avaliada. Eficiencia. Não será feito testes de performance ou algo parecido Manutenabilidade Será avaliado. Portabilidade. Será avalidado

3.3 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) elaborou 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. 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 a classe tem métodos espefíficos 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.

Todas essas métricas tem uma relação inerentes 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.

Fazer referências de estudos relacionando CKsuite e Manutenabilidade.

quanto maior dit maior reuso mas é melhor usar composição.

3.4 Model View Controller

O padrão Model View Controller surgiu como uma solução genérica para usuários de uma sistema de planejamento manipulem dados complexosReenskaug (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 "... was to support the user's mental model of the relevant information space and to enable the user to inspect and edit this information.". 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) descreve a estrutura do MVC onde a view tem seu controller exclusivo mantendo uma dependência cíclica entre ambos e 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 mais 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) "Of these the separation of presentation from model is one of the most fundamental heuristics of good software design".

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.

Gamma et al. (1995) cita Krasner e Pope (1988) fazendo uma análise dos objetos que compõem o MVC relacionando-os com outros padrões de proejtos descritos em seu catálogo. O desacoplamento entre a view e o model e a propagação das mudanças de estado no model para os objetos regitrados como dependentes do model pode ser descrico como uma implementação do padrão Observer. A hierarquia de views é um exemplos de Composite pois uma view pode ser constituída por sub-views para compor views complexas. O Strategy é aplicando ao controller que encapsula o algoritmo que vai alterar a view e o model podendo ser substituído por uma outra implementação que deixa de responder às interações com o usuário.

Segundo Krasner e Pope (1988) o Model "...can be as simple as an integer (as the model of a counter) or string (as the model of a text editor), or it can be a complex object". O model pode ser implementado usando o pardrão Facade para simplificar as interações com o model dependendo da complexidade do domínio que ele representa.

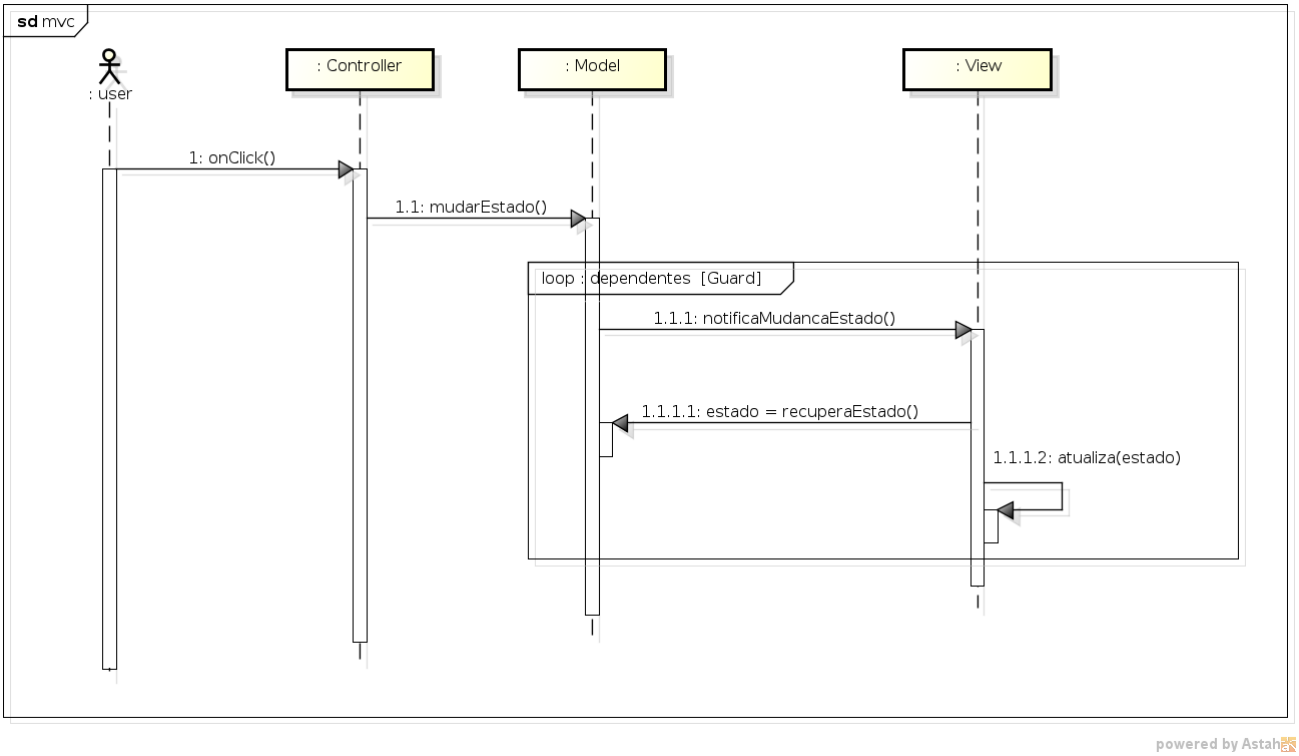

Figura 2 – Diagrama de Sequência do MVC/Fonte: Próprio Autor

3.5 Model View Presenter

O MVP é um modelo de programação para implementação de interfaces com o usuário desevolvido 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[1] adotando os conceitos do MVP onde salienta que a maioria dos sistemas operacionais com ambiente gráfico fornece um conjunto de componetes (Widgets) no qual está contido a responsbilidade do controller.A maior parte do comportamento do caso com o usuário é implementada no presenter que está diretamente associado a 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, permitindo uma maior cobertura de testes aplicados ao presenter. Dessa forma o model não se comunica com a view por meio do observer pattern, sendo que a view séra atualizada pelo presenter.

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.

Abordar As variações do MVP.

3.6 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 dispositivos smartphones e tablet que é o foco deste trabalho. A plataforma é contituída por uma pilha softwares e frameworks tendo em sua base o sistema operacional e seu drivers seguido da máquina virtual que executa os aplicativos android e bibliotecas auxiliares, SDK e ferramentas de desenvolvimento e aplicativos básicos.

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

Activity Representa uma atividade que o usuário executa no aplicativo em um determinado momento. Usado para criar a interface contém todos os componetes visuais e responde à interações do usuário.

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

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

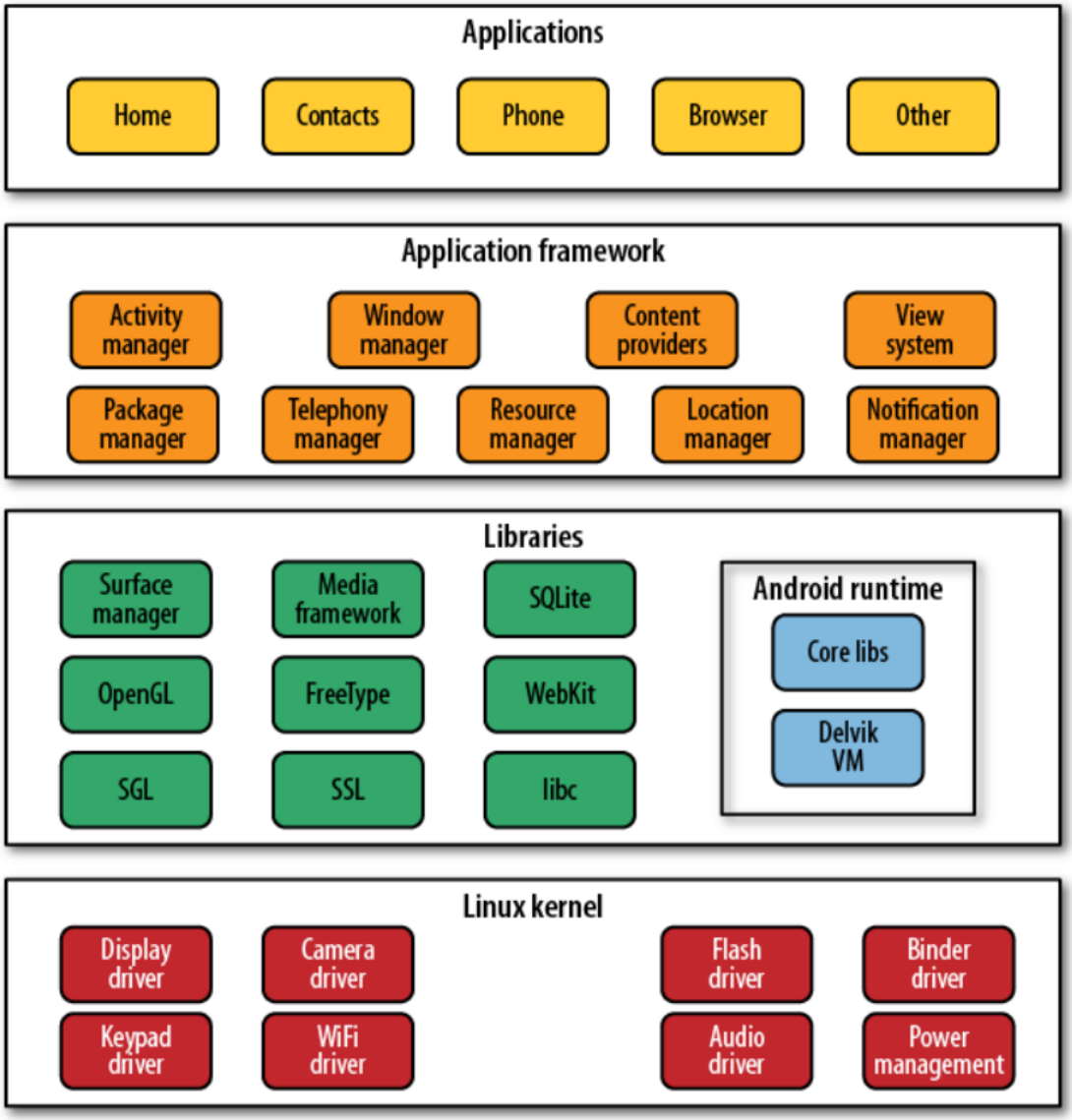

Figura 3 – Android Stack/Fonte: Learning Android

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 implemntação a ser usada para análise da qualidade dessa arquitetura.

A activity terá a responsabilidade da view pois é através dela 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. Essa falta de coesão da classe activity se deve ao uso abusivo de herança e que pode causar confusão com relação a responsabilidade da implementação de uma activity no aplicativo. Segundo Reenskaug (1979) The View and Controller roles may be played by the same object when they are very tightly coupled. Example: 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 por ser uma classe auxiliar à view pode ser implementada como uma classe java simples. O Model será representado por componetes DAO (Data Access Object) para armazenamento e recuperação dos dados e classes auxiliares que encapsulam regras relacionados ao domínio de negócio.

Para concluir a seção sobre MVC e MVP destacar que A principal características desses padrões de projetos é que ele promover maior coesão nos citar Tom deMarco

baseado em outras pesquisa será aplicado padões de projetos para melhorar a qualidade pegar referências, relacionar Coesão = Qualidade.

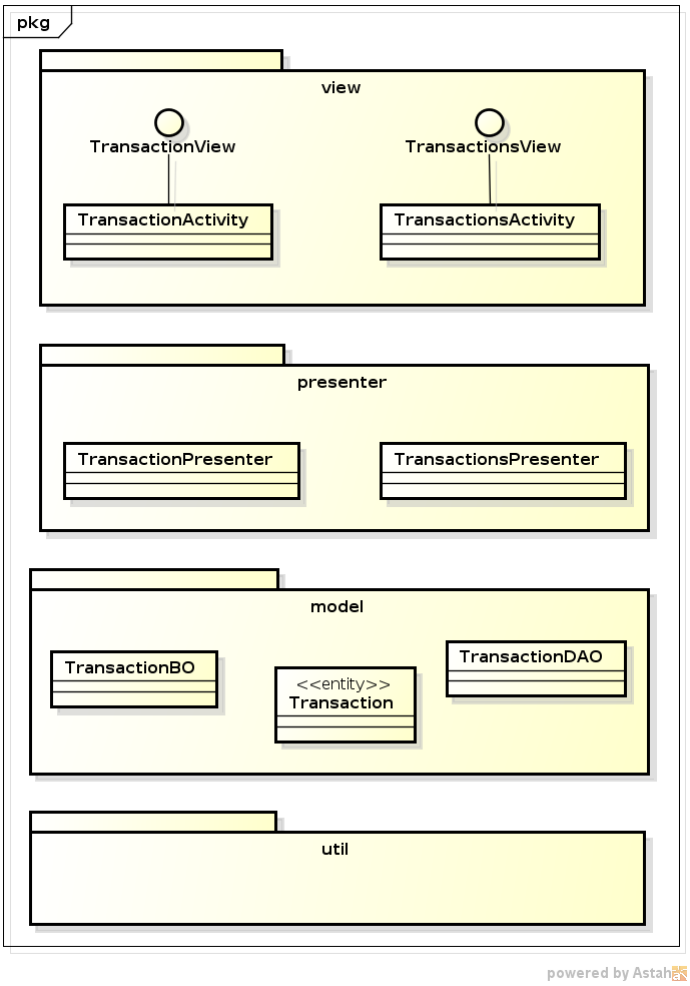

Figura 4 – Arquitetura POC1/Fonte: Próprio Autor

4 Execução da Pesquisa

4.1 Ferramenta para coleta: CK metrics for Java

Porque essa ferramenta? Como ele calcula? como interpretar os resultados fornecidos? Referencia na literatura dessa ferramenta.

este artigo mostra o resultado de várias ferramentas e analisa seus resultados Comparing Software Metrics Tools

4.2 Análise do objeto de estudo

Levantamento a arquitetura atual do projeto e seleção do módulo a ser refatorado.

Será mantida a interface publica dos componentes refatorados, para não afetar outras partes do aplicativo.

Ambiente de desenvolvimento, Gerenciamento de configração.

4.3 Primeira Fase: Coleta de métricas para linha de base

4.4 Segunda Fase: Refatoração e coleta de métricas para comparação

Explicar como foi aplicado, o que foi alterado, problemas no desenvolvimento, explicar decisões de design. etc.

5 Resultados e Conclusões

5.1 Conclusões

É um equívoco MVC no contexto atual pois o padrão não se adequa. o que existe são evoluções aplicadas a uma determinada situação desenvolvimento web, desktop e

abordar quais seriam os parâmetros aceitáveis para as métricas(difícil afirmar quais são esses parametros, pois cada ferramenta de coleta tem sua propria forma de calcular. citar artigo sobre isso)

Validade dessas métricas, talvez o ideal fosse customizar essas métricas para o framework?;

Possível aumento do código com essa divisão com mais classes — Isso ocorreu

O presenter deveria ser independente de tecnologia mas na plataforma android, seguir isso a risca iria criar uma aberração pois a activity ou fragmentos proveem muitos recursos usados para diversos interesses(View Controller/Presenter Model), por isso é tão natural esses componentes serem implementados com diversas responsabilidades.

O componente Loader é dificil de refatorar, pois esta muito acoplado com a view!

O foco da refatoração foi remover o model e o estado da view para o presenter

Quais metricas foram afetadas.(todas exceto NOC e DIT)? - WMC,CBO,RPC,LCOM foram afetadas

a refatoração melhorou cbo/lcom e piorou ou manteve rpc wmc,

WMC 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 inteface, portanto implementam uma interação com o usuário bem específica. não havera impacto negativo, além disso a quantidade de métodos aumentou mas com uma complexidade menor pois métodos com várias estruturas de controle forem quebrados em métodos menores, podendo ser avaliado como positivo.

Houve 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.

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: MVVM, MVP-VM, MVPC.

Desenvolver métricas

Este trabalho não fez uma avaliação dos impactos na performance do aplicativo devido ao uso desses padrões. A inclusão de mais objetos interagindo, redirecionando mensagens pode depreciar a performance.

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 16.

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 23.

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 na página 16.

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 2 vezes nas páginas 13 e 14.

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 2 vezes nas páginas 15 e 16.

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 18.

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 6.

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.