

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

<b>1</b>	<b>Introdução</b>	<b>5</b>
1.1	Motivação	5
1.2	Problematização	5
1.3	Hipótese	6
1.4	Objetivos	6
1.5	Trabalhos Relacionados	6
1.6	Contribuições	7
1.7	Organização do Trabalho	7
<b>2</b>	<b>Metodologia</b>	<b>9</b>
2.1	Objeto de Estudo	9
2.2	Processo de Experimentos	10
2.3	Ferramentas	10
<b>3</b>	<b>Referencial Teórico</b>	<b>11</b>
3.1	Princípios e Padrões de Projetos	11
3.2	Model View Controller	12
3.3	Model View Presenter	14
3.4	Framework Android	15
3.5	Refatoração do Objeto de estudo	17
<b>4</b>	<b>Resultados e Conclusões</b>	<b>19</b>
4.1	Trabalhos Futuros	19
4.2	Conclusões	19
	<b>Referências</b>	<b>21</b>



# 1 Introdução

A tendência do mercado de dispositivos handset é aumentar segundo IDC é esperado 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 demanda 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 manutenibilidade, 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 manutenibilidade 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 projetos 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 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 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 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.
- 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.

## 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 dentro framework android.

A diversidade de projetos de software torna difícil inferir se as métricas podem ser consideradas um parâmetro 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, definição do projeto, experimentos, conclusões.





## 2 Metodologia

Esta pesquisa se caracteriza 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 componentes do android que podem assumir as responsabilidades definidas nos padrões e definir como implementá-los.

Para validar a hipótese apresentada neste trabalho a pesquisa terá uma abordagem quantitativa através da análise de dados estatísticos de métricas que expressam 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, relacionadas 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<sup>1</sup> que é um dos aplicativos básicos pré-instalados com o sistema operacional. Este projeto é open source mantido pelo The Android Open Source Project 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 a partir 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

<sup>1</sup> [https://github.com/android/platform\\_packages\\_apps\\_contacts](https://github.com/android/platform_packages_apps_contacts) (PMD25073)

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

## 2.2 Processo de Experimentos

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

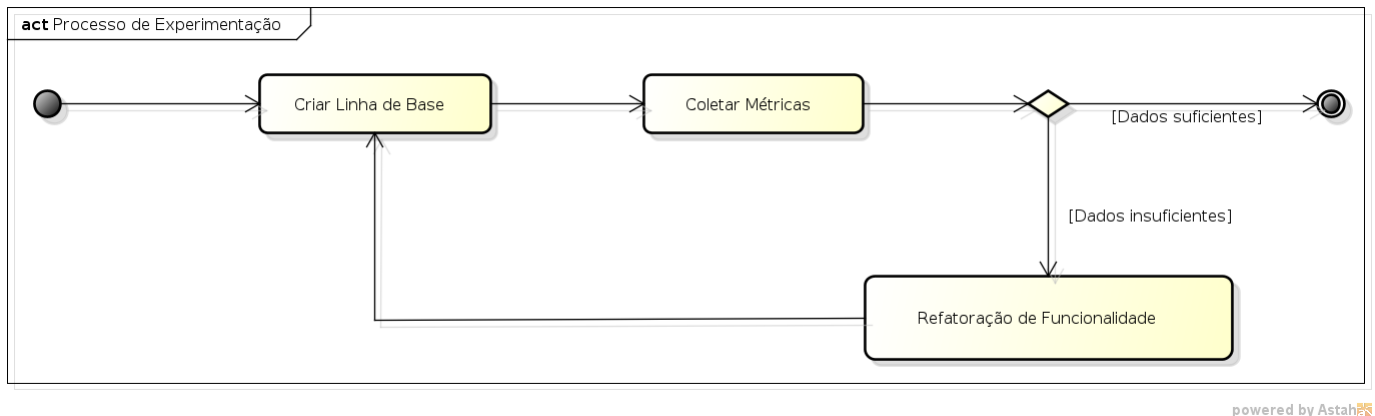


Figura 1 – Processo de Experimentação Fonte: Próprio Autor

**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 ferramenta 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<sup>2</sup>.

<sup>2</sup> <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 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 apresenta características fundamentais. Coesão, acoplamento etc (DEMARCO, 1979), (PAGE-JONES, 1988).

Um padrão dentro do contexto de estudo deste presente trabalho pode ser definido como uma técnica efetiva cuja a sua aplicabilidade é aceita e difundida 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 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.

Com o intuito de documentar os padrões de projetos de forma objetiva para facilitar o aprendizado dos mesmos, a comparação e escolha do padrão a ser aplicado, Gamma et al. (1995) elabora um modelo que descreve as seguintes características para cada padrão:

**Nome e Classificação**

**Intenção**

**Também conhecido como**

**Motivação**

**Aplicabilidade**

## Estrutura

### Participantes

**Collaborations** How the participants collaborate to carry out their responsibilities.

**Consequences** How does the pattern support its objectives? What are the trade-offs and results of using the pattern? What aspect of system structure does it let you vary independently?

**Implementation** What pitfalls, hints, or techniques should you be aware of when implementing the pattern? Are there language-specific issues?

**Sample Code** Code fragments that illustrate how you might implement the pattern in C++ or Smalltalk.

**Known Uses** Examples of the pattern found in real systems. We include at least two examples from different domains.

### Related Patterns

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 os padrões de projeto promovem maior coesão, melhor separação de interesses e baixo acoplamento no sistema.

## 3.2 Model View Controller

O padrão Model View Controller surgiu como uma solução genérica para 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 “...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 domínio 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 representação computacional de seu modelo mental três componentes são definidos:

**Models** - É o componente 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 toda 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 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 características específicas 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 estender. A Figura 2 esclarece a interação entre os componentes.

Gamma et al. (1995) cita Krasner e Pope (1988) fazendo uma análise dos objetos que compõem o MVC relacionando-os com outros padrões de projetos. O desacoplamento entre a view e o model e a propagação das mudanças de estado no model para os objetos registrados como dependentes do model pode ser descrito como uma implementação do padrão Observer. A hierarquia de views é um exemplo de Composite pois uma view pode ser constituída por sub-views para compor views complexas. O Strategy é aplicado 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.

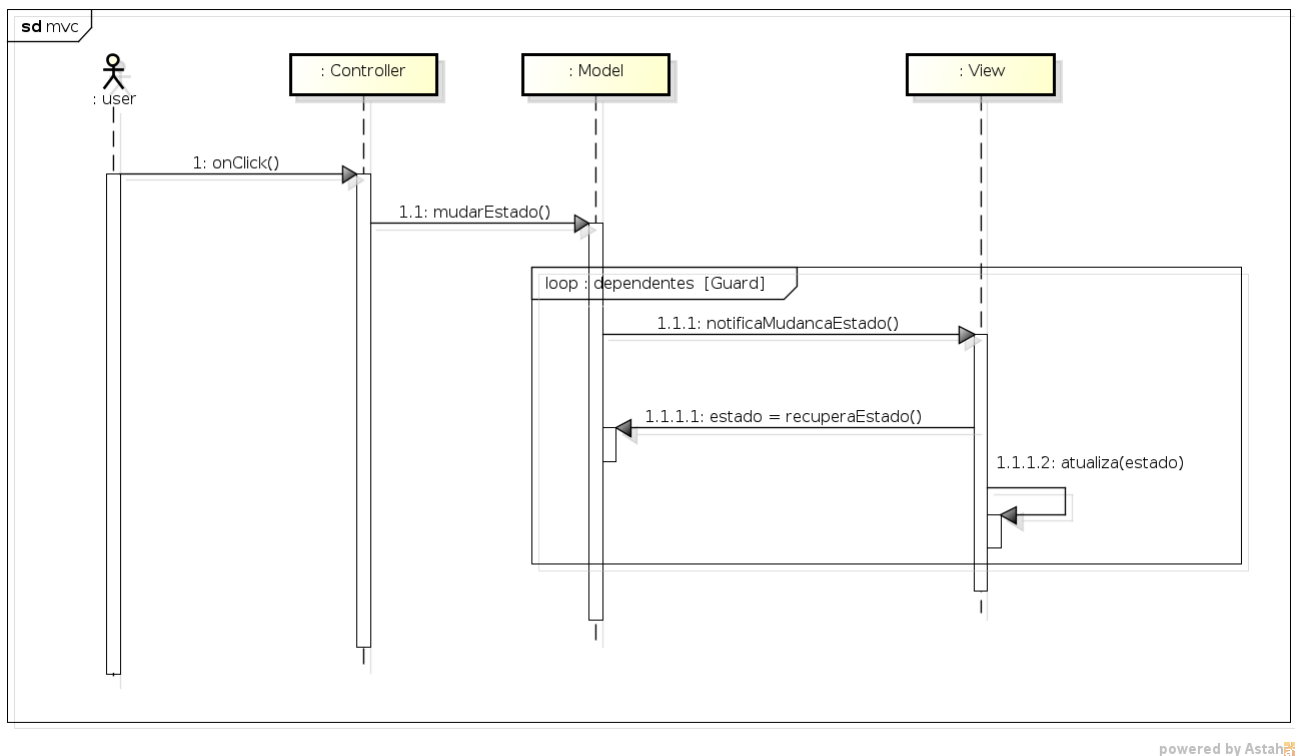


Figura 2 – Diagrama de Sequência do MVC/Fonte: Próprio Autor

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 padrão Facade para simplificar as interações com o model dependendo da complexidade do domínio que ele representa.

### 3.3 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 chamada 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:

- Model
- Selections
- Commands
- View

- Interactor
- Presenter

Os conceitos do MVP são descritos em [Potel \(1996\)](#) de forma genérica permitindo interpretações para uma implementação efetiva. [Bower e McGlashan \(2000\)](#) descreve a implementação de um framework para Dolphin Smalltalk<sup>1</sup> dotando os conceitos do MVP.

O MVC foi concebido quando as interfaces com o usuário não eram tão ricas. [Bower e McGlashan \(2000\)](#) 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 da interface com o usuário é implementada no presenter que está diretamente associado a View [Bower e McGlashan \(2000\)](#)

discorrer sobre a interpretação do fowler:Passive View

discorrer sobre a interpretação de An Architecture and Implement Model for Model-View-Presenter Pattern

O modelo de programação MVP define mais responsabilidades para o presenter em relação ao controller do MVC pois o presenter redireciona as interações do usuário para executar lógica de negócio a ser aplicada sobre o model e organiza como os outros componentes irão funcionar.

## 3.4 Framework Android

Historia características Arquitetura.

Componentes Activity service broadcastReceiver ContentProvider

Como DP resolvem problemas, seguir linha de raciocínio para aplicar padrões em android.

Análise do componente Activity, UI Thread, hierarquia de herança e algumas funcionalidades que dependem da activity e como isso interfere na aplicação do padrão (Por experiência confirmo que é negativa). mostrar exemplo de uma view em lista para smartphone e outra para tablet com grid usando o mesmo modelo

Experimentos

activity - controller+async task - listeners, activity - controller+async task - observer pattern, activity - controller+async task - localbroadcast activity - controller+async task - Messages Activity+Fragment

[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 uma boa análise

<sup>1</sup> Implementação da Linguagem de programação Smalltalk, <http://www.object-arts.com>

do problema em questão para decidir o nível de granularidade que esses componentes podem ter portanto é recomendável manter sempre essa separação.É aplicável para



## 3.5 Refatoração do Objeto de estudo

Com base na literatura revisada esta seção tem como objetivo fazer uma análise dos componentes do framework android e projetar uma camada de apresentação utilizando o padrão MVP. Será realizado uma análise tendo como objetivo definir usada um referência de implemntação a ser aplicada na refatoração(Referências sobre Refatoração de Código) Quem vai ser o model?

OnTouchListener pode exercer papel de controller pois pode ser usado para interpretar os gestos do usuário e direcioar para o model.

Na definição clássica do padrão MVC é possível identificar a implementação do padrão Observer usado para a comunicação entre o model e a view.

Observer=Lapsed listener problem=memory leak e bugs porque uma activity pode ser ou estar sendo referenciada por um listener que se não for desregistrado vai causar problema

Destacar problema com o observer pattern e pra solucionar isso usar publish–subscribe messaging pattern

O Broadcast Receiver é uma implementação do padrão pubsub onde diversos subscribers se registram para receber mensagens(intents) de seu interesse.

Padrões Criacionais. O objeto Application é um singleton, a activity e fragmentos são criados pelo android e fica difícil aplicar padrões criacionais é possível identificar o padrão templete method nos ciclos de vida desses componentes.

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.



## 4 Resultados e Conclusões

### 4.1 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.

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.

### 4.2 Conclusões

Validade dessas métricas, talvez o ideal fosse customizar essas métricas para o framework. Possível aumento do código.



# 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 na página 9.
- DEMARCO, T. *Structured Analysis and System Specification*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1979. ISBN 0138543801. Citado na página 11.
- 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 13.
- 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 13.
- 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 12, 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.
- PAGE-JONES, M. *The practical guide to structured systems design*. 2. ed. ed. London: Prentice-Hall International, 1988. (Yourdon Press Computing Series). ISBN 0136907776. Citado na página 11.
- POTEL, M. *MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java*; 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 planning system*. [S.l.], 1979. Acesso em: 1.8.2013. Citado 2 vezes nas páginas 12 e 15.
- 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.
- 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.