ThoughtWorks®

ESCREVA APPS DE QUALIDADE EM QUALQUER ARQUITETURA

QUEM SOMOS

Daniel Debastiani

Consultor Mobile

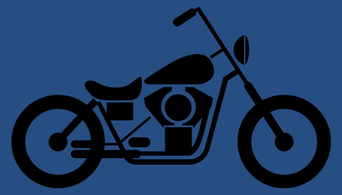
Guitarrista

Guilherme Cherubini

Consultor Mobile

Híbrido

@gcherubini

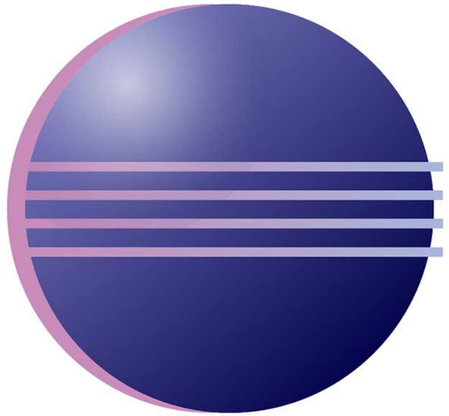


ThoughtWorks®

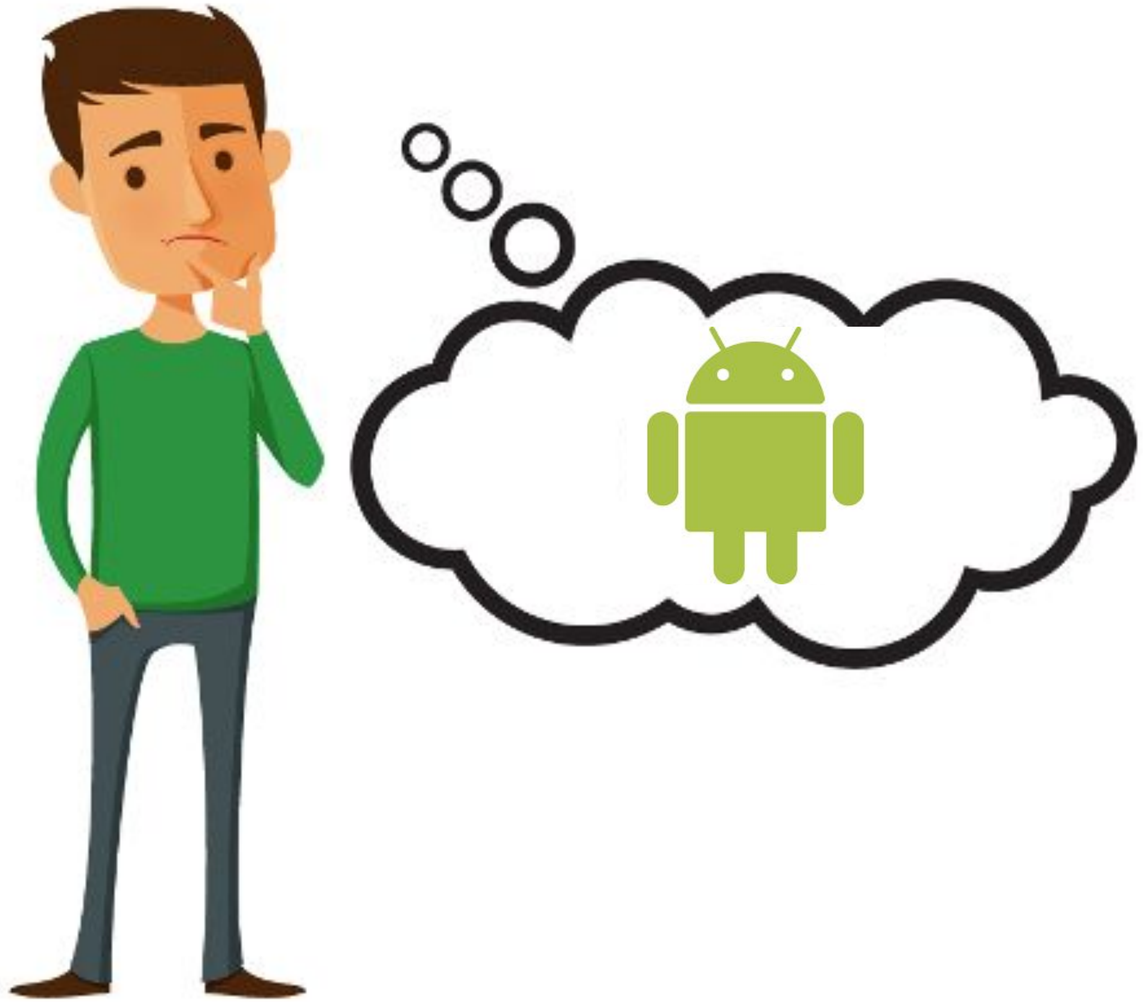
ThoughtWorks®

NO INÍCIO TUDO ERA MATO!!

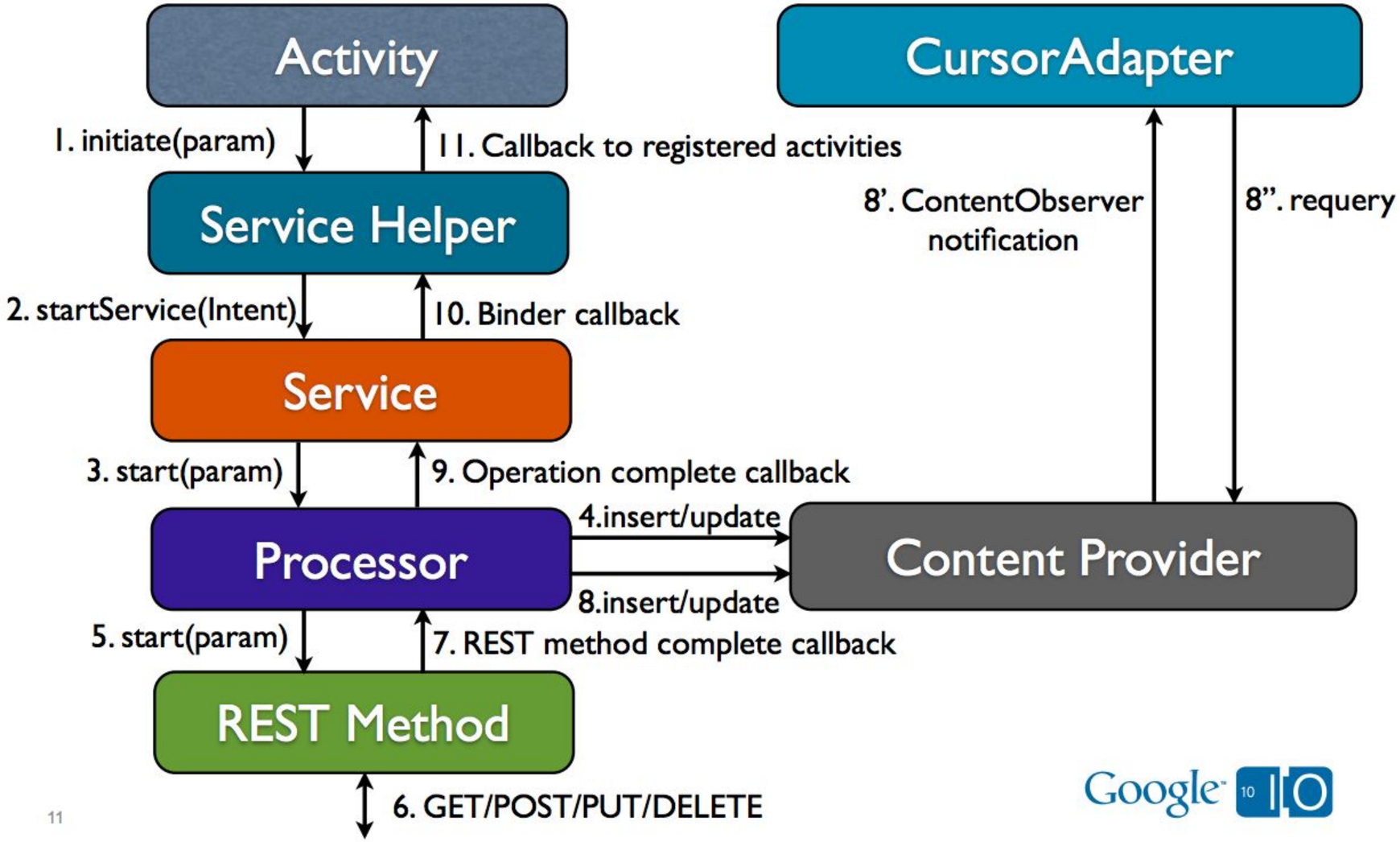
O INÍCIO: Eclipse + ADT



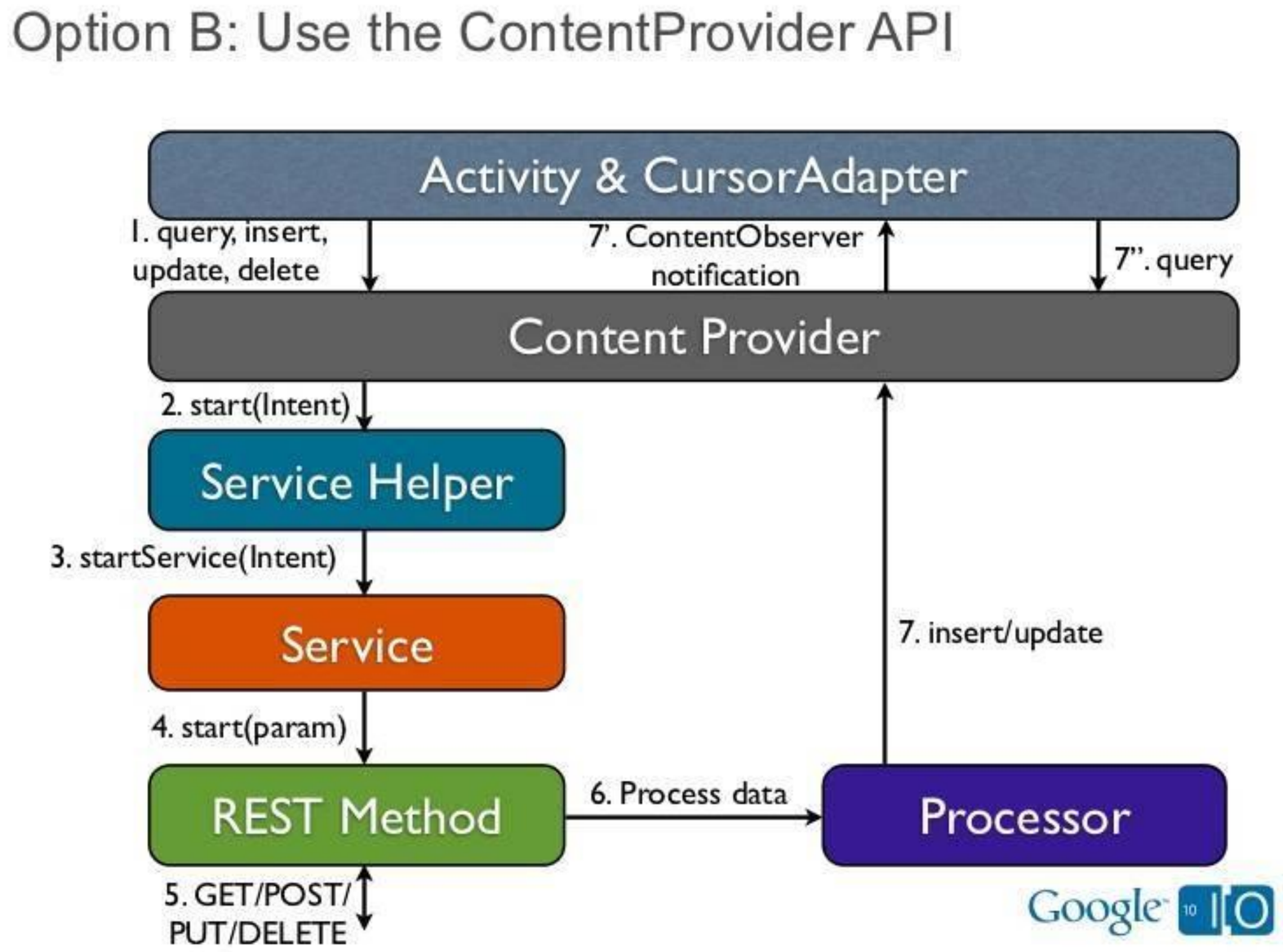
O INÍCIO: Como estruturar um app Android?



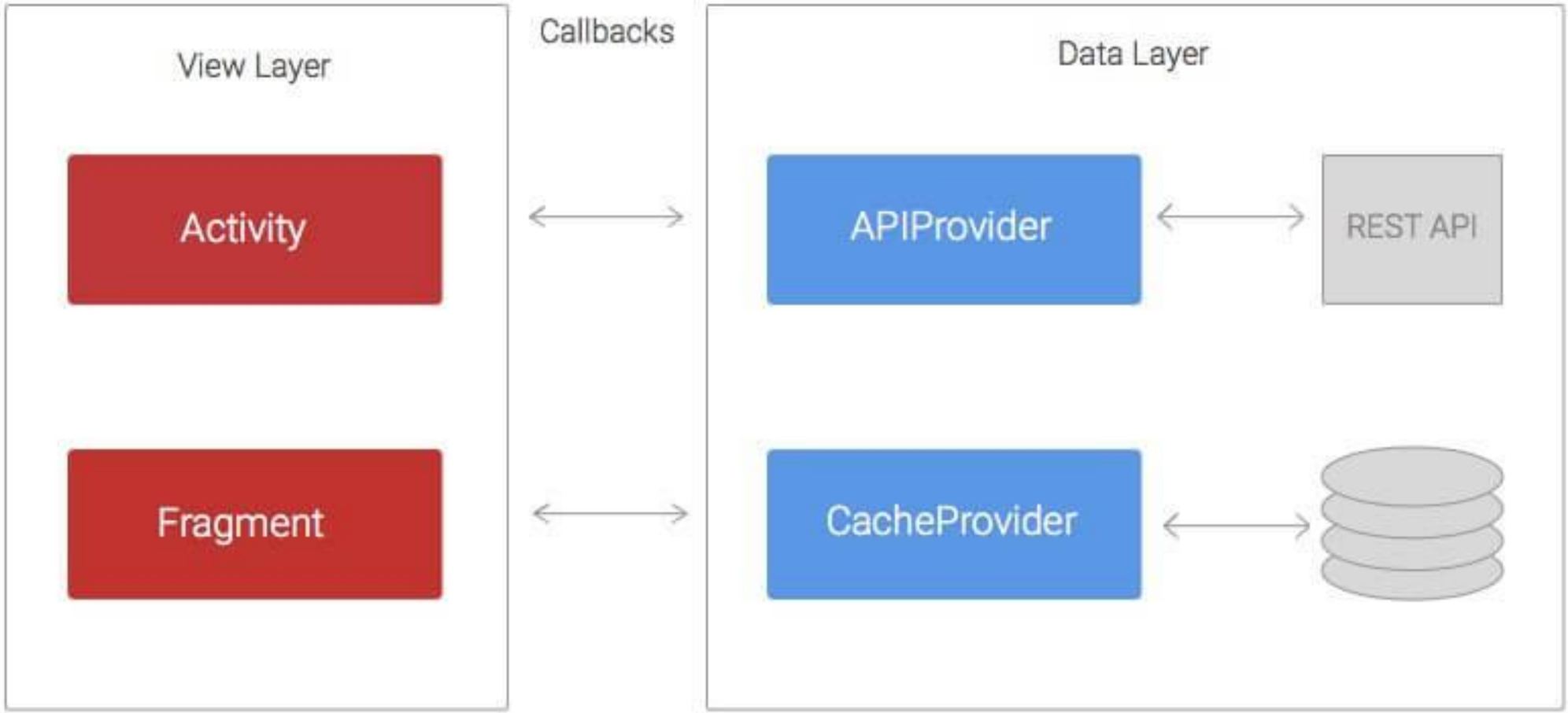
O INÍCIO: Content Provider + Service



O INÍCIO: Content Provider + Service



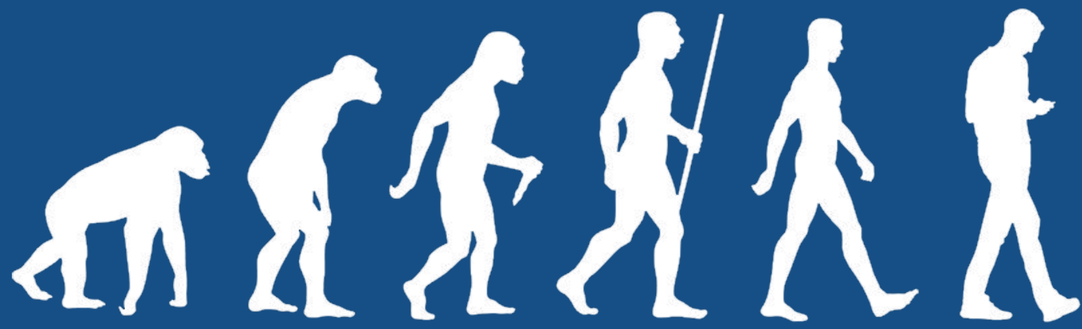
O INÍCIO: MVC?


View Layer
Activity
Fragment
Callbacks
Data Layer
APIProvider
REST API
CacheProvider

ALGUNS SMELLS COMEÇAVAM A SURGIR..

- Não havia muita preocupação com separação de responsabilidades e camadas
- Por simplicidade, tudo vai na Activity/Fragment
- Chamadas de Network (API) e banco de dados espalhadas pelo código.
- Mapear JSONs da API como Models de Banco; Persistir diretamente retornos de API.

ThoughtWorks®



A COMUNIDADE COMEÇA A EVOLUÍR

PADRÕES SURGEM

- MVC
- MVP
- Databinding
- MVVM


omfg!
mvi
mvw
ribs
mvrx
viper
mvc
mvvm
mvp
mva
redux

ThoughtWorks®

COM TANTA OPÇÃO, O QUE EU FAÇO?

SURGEM OS QUESTIONAMENTOS

- MVP é inviável?
- MVVM resulta em ViewModels gigantes e não manuteníveis?
- VIPER é uma solução complexa e overkill?

ThoughtWorks®

CULPANDO OS PADRÕES

“

Model-View-ViewModel **é muito melhor que** MVP

— Internet.

”

“

Viper é clean e **funciona melhor**

— Internet.

”

Eu gosto que a View seja burra.

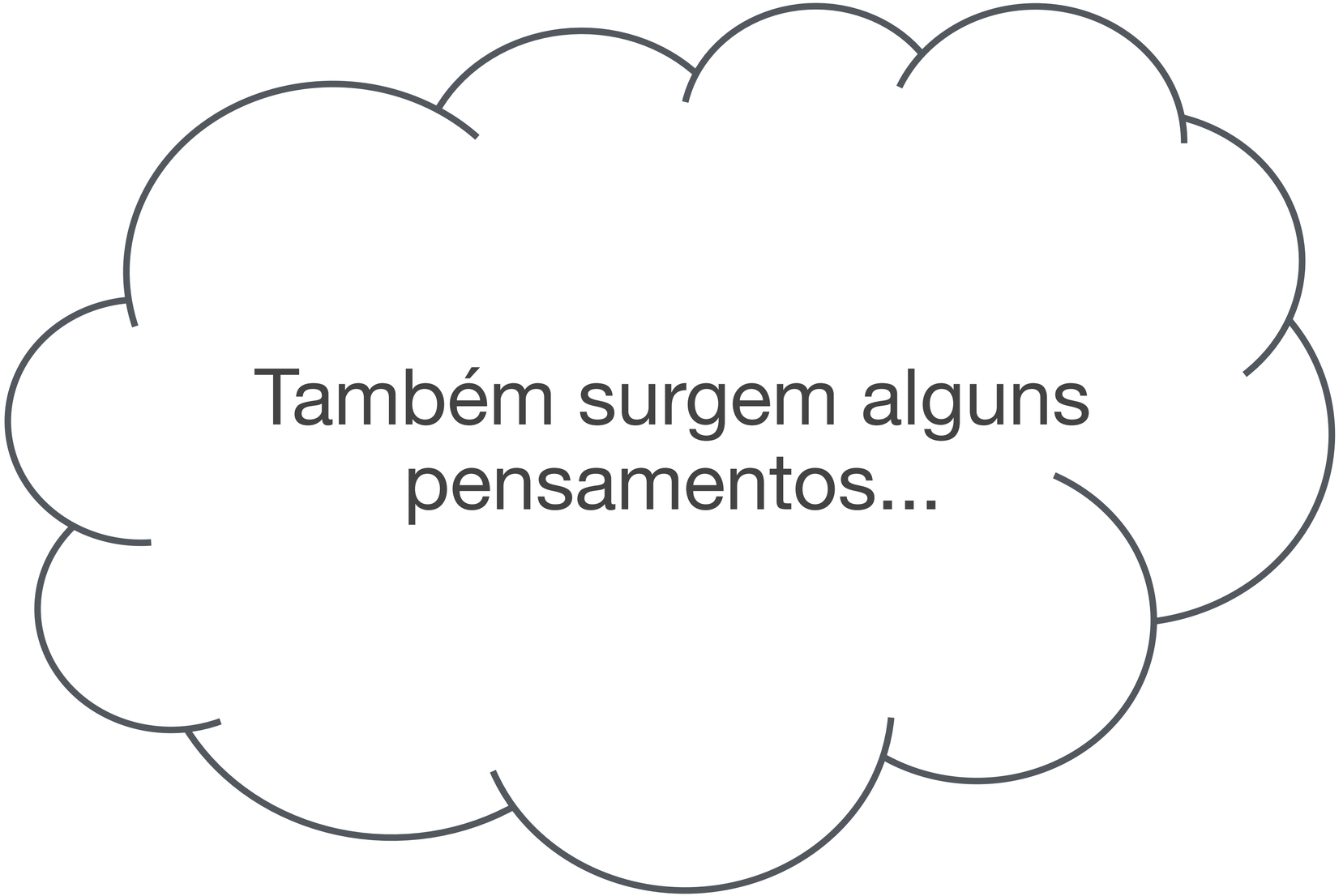
— Internet.

Eu prefiro que isso seja feito em outra camada.

— Internet.

NÃO IMPORTA O PADRÃO, SE O CÓDIGO...

- Bagunçado
- Complicado
- Não legível
- Não testável
- Não evolutivo


Também surgem alguns
pensamentos...

“Meu app é MVP. Será que posso usar um único Presenter para buscar dados da API e gravar no Banco? Ou preciso de dois Presenters?”

“Será que posso utilizar o mesmo ViewModel para dois Fragments?”

“Aonde eu coloco o tracker do Analytics? ViewModel, View ou Interactor?”

VAMOS CULPAR O PADRÃO!

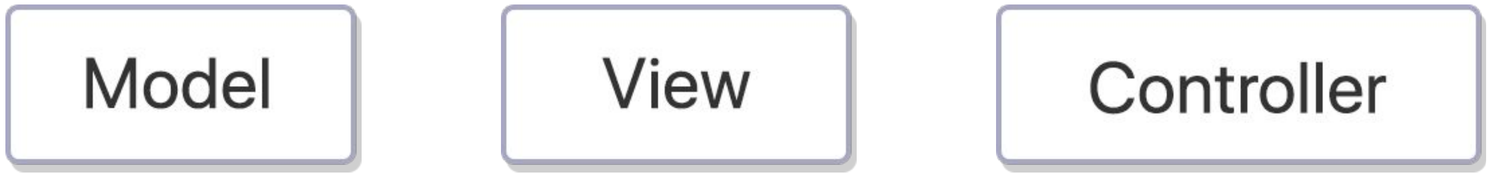
NÃO.

Seu app **não é** em MVP ou MVVM, ele **utiliza o** MVP ou MVVM.

ThoughtWorks®

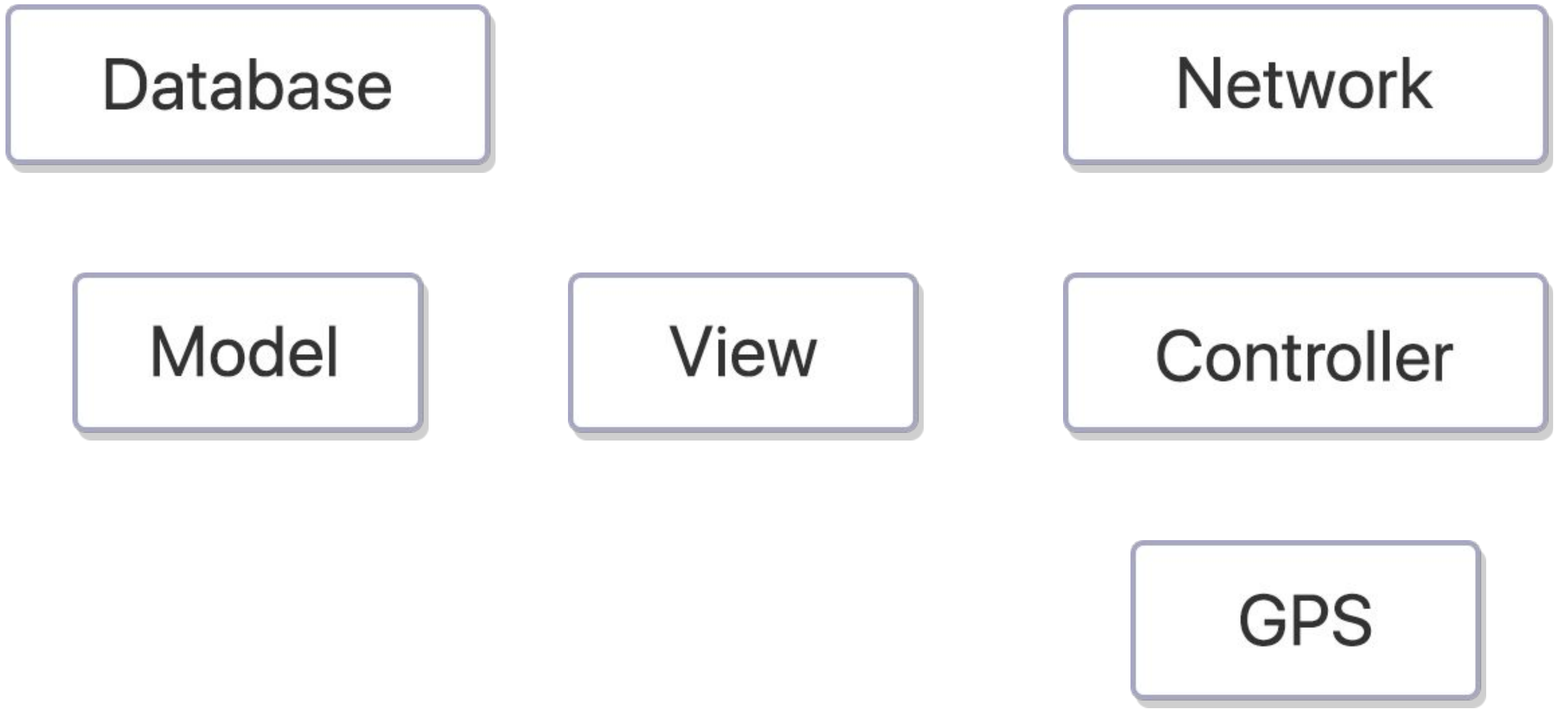
NÃO COLOQUE TUDO DENTRO DE CAIXINHAS

MVC

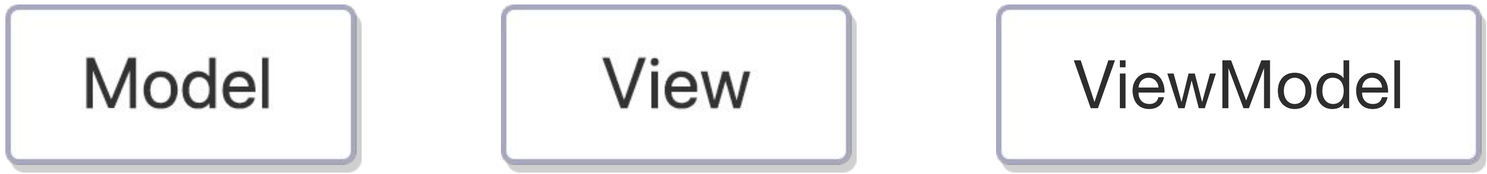


COLOCANDO TUDO DENTRO DE CAIXINHAS

ESTRUTURANDO AO REDOR

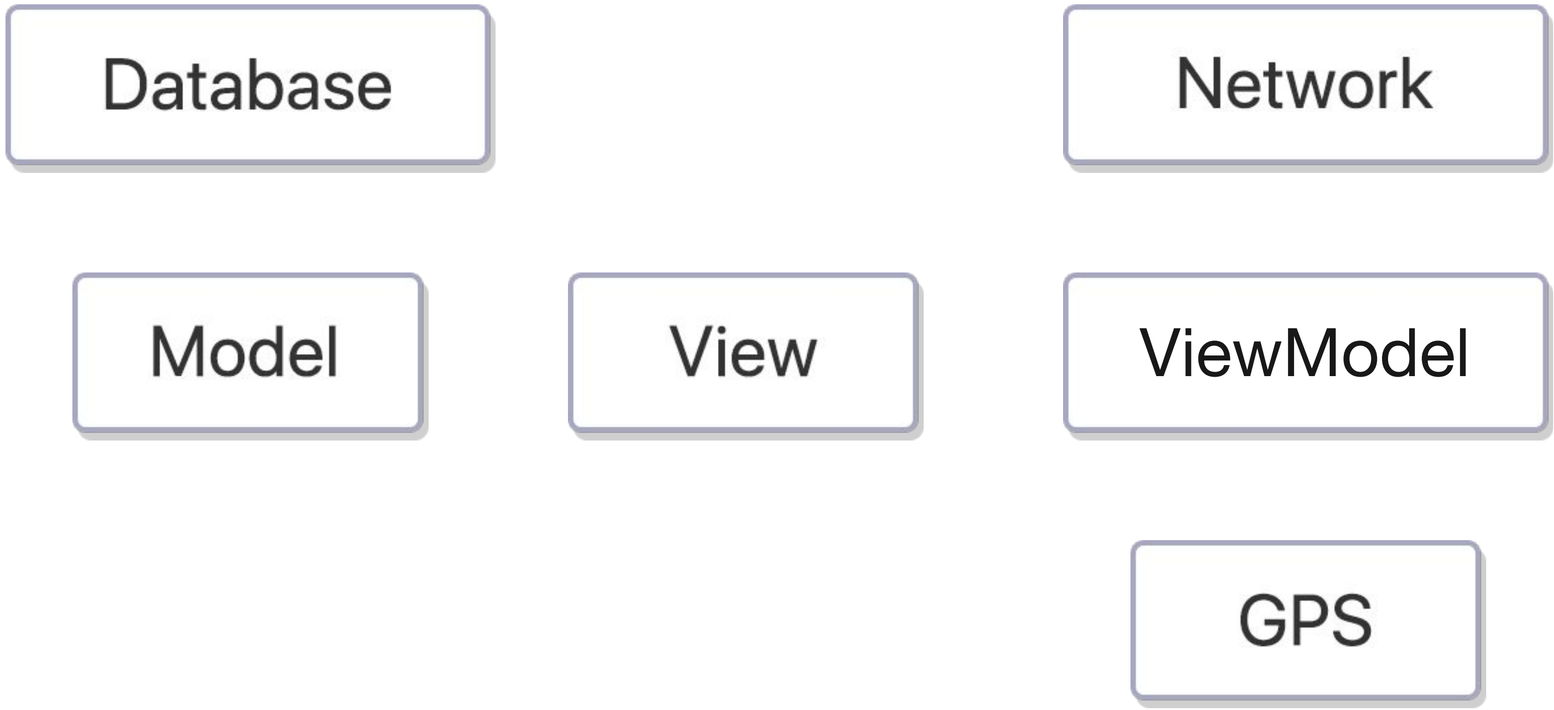


MVVM



COLOCANDO TUDO DENTRO DE CAIXINHAS

ESTRUTURANDO AO REDOR



ThoughtWorks®

CHEGA DE FIXAÇÃO: PADRÕES SÃO PADRÕES

PADRÕES NÃO SÃO RECEITAS

- O padrão arquitetural se adapta a seu app, não o contrário
- São sugestões, uma ajuda, um guia, uma direção
- É a solução para problemas em comum que vários desenvolvedores tiveram e foram obtidos por reuso contínuo, observação e refinamento
- Não são uma solução completa
- Não são uma religião

PADRÕES NÃO GARANTEM A QUALIDADE

- Não garantem código limpo
- Não garantem boas práticas de desenvolvimento
- Não garantem testes
- Não vão te salvar de bugs recorrentes ou software não-resiliente

O QUE AJUDARIA ENTÃO?

- Bons desenvolvedores
- Boas práticas
- Acordos entre o time
- Processos de qualidade de software
- Onboarding
- ...

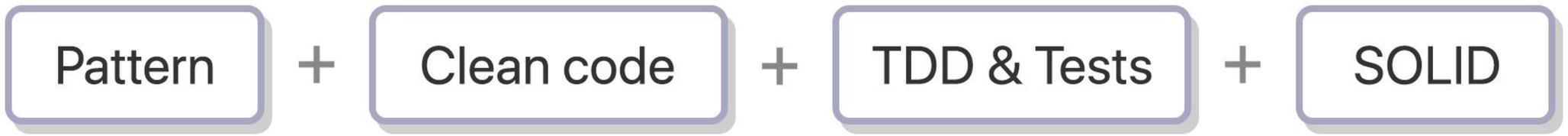
ThoughtWorks®

UMA NOVA FORMA DE PENSAR

UMA NOVA FORMA DE PENSAR

- Um novo padrão
- Um novo mindset
- Pilares que você e seu time precisam concordar
- Um pacto

PACTS

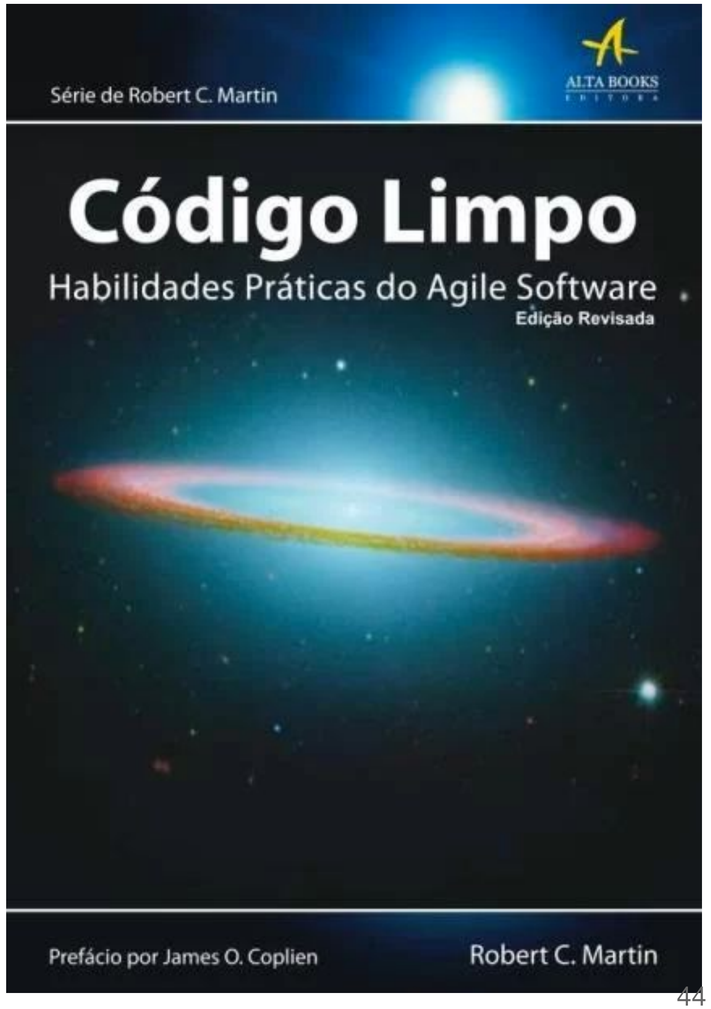

Pattern
+
Clean code
+
TDD & Tests
+
SOLID

Pattern + Clean code + TDD & Tests + SOLID

- MVP
- MVVM
- VIPER
- MVI
- O que você se sentir mais confortável!

Pattern + **Clean code** + TDD & Tests + SOLID

- Escreva um código elegante e simples
- Feito para humanos entenderem
- Que você mesmo vá entender no futuro



44

Pattern + Clean code + **TDD & Tests** + SOLID

- Por que testes?
  - Evita e ajuda a identificar bugs
  - Salvam tempo e dinheiro
  - Aumenta a manutenabilidade do software
  - Traz mais segurança no refactoring
  - Documentação viva do software

- Por que TDD?
  - Foco em pedaços pequenos de funcionalidade ➡️ cobertura de qualidade
  - Melhores interfaces
  - Garante que seu software atende as regras de negócio do sistema
  - Melhora a arquitetura
- **Código bem testado é bom**

- 5 princípios:
  - Single responsibility principle
  - Open/closed principle
  - Liskov substitution principle
  - Interface segregation principle
  - Dependency inversion principle

- Permite:
  - Design compreensível
  - Design flexível
  - Design manutenível
  - Design extendível

Pattern + Clean code + TDD & Tests + SOLID

QUAL É A MELHOR ARQUITETURA?

E QUAL A MELHOR ARQUITETURA?

- Depende do seu negócio
- Do seu momento
- Capacidade técnica
- Contexto

É HORA DE FAZER ESSE PACTO!

OBRIGADO!

DANIEL DEBASTIANI

@DSDEBASTIANI

GUILHERME CHERUBINI

@GCHERUBINI

LEANDROALONSO.COM

@LEANDROW

LINKS DE APOIO:

Sobre a Palestra:

- Write quality mobile apps in any architecture - https://thght.works/2GXC255
- Riblets: https://ubr.to/2VGFtFy
- MvRX: http://bit.ly/2UVT0EG
- Discover: https://tinde.rs/2Wilhpr

Sobre a Thoughtworks:

- TechRadar: https://thght.works/2V8f0Be
- Enegrecer a Tecnologia: https://thght.works/2XVnQ2e
- Programa Desenvolve: https://thght.works/2LkXy8b
- Vagas TW Global: https://thght.works/2VGIvJY