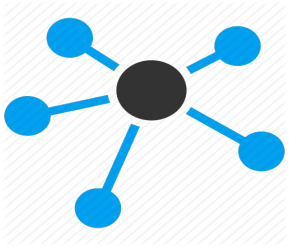
iOS架构设计

张应龙

2016-04-21

架构设计目的

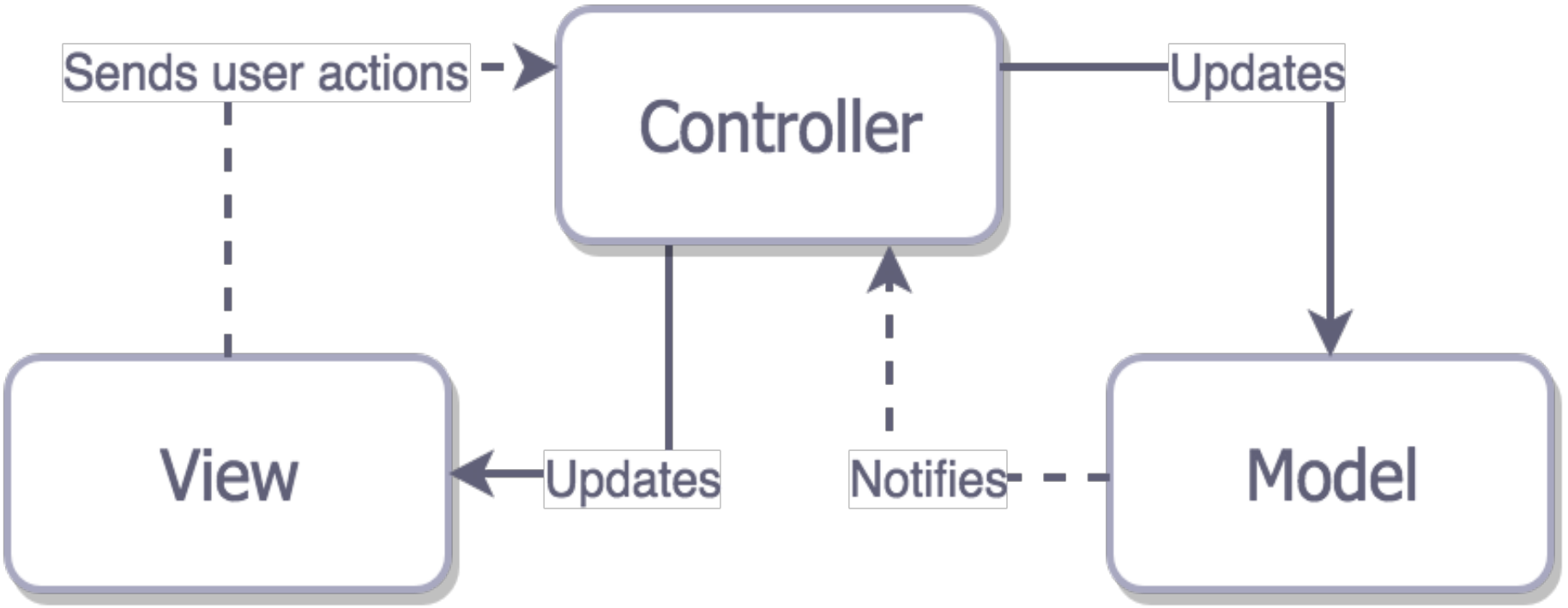
- 单一职责原则



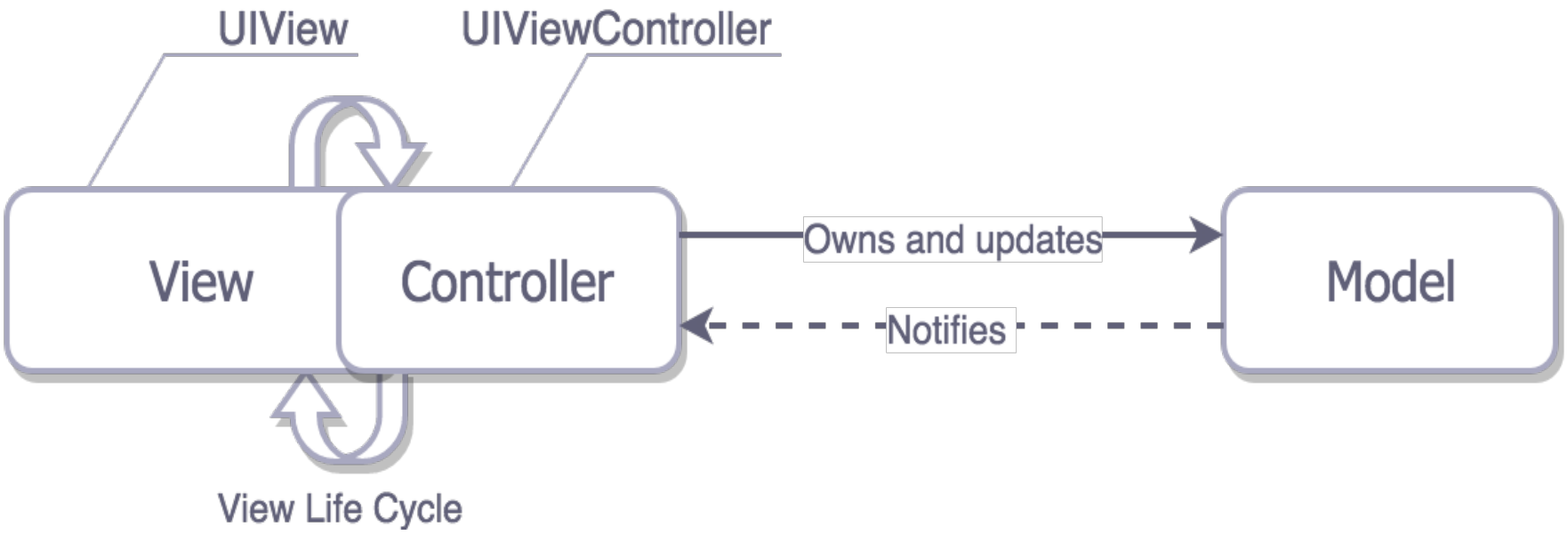
- 可测试原则

- 易用原则

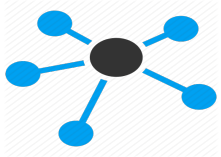
Apple MVC


Sends user actions
Controller
Updates
View
Updates
Notifies
Model

实际的MVC


UIView
UIViewController
View
Controller
Owns and updates
Notifies
Model
View Life Cycle

MVC的评价

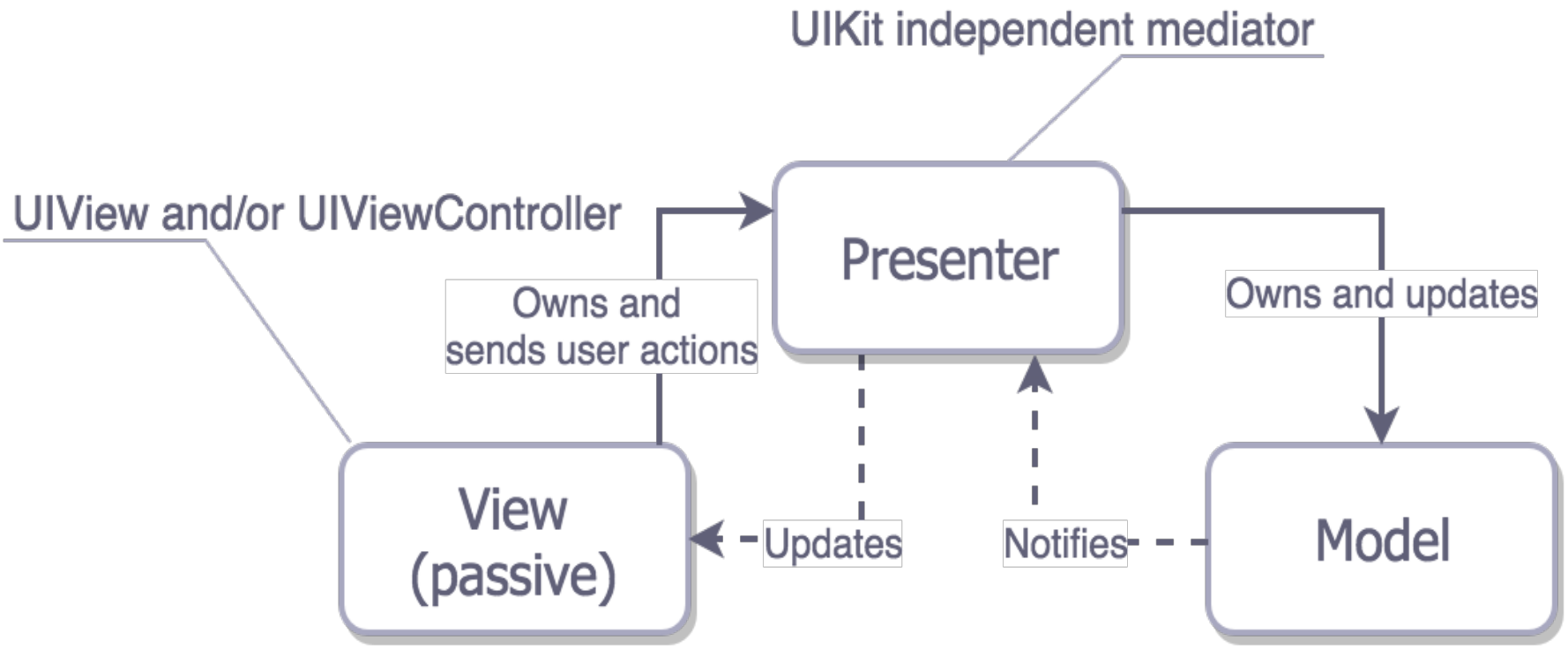


- 职责划很差

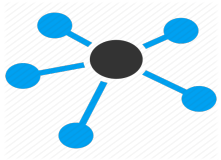
- 可测性很差

- 易学易用

Google MVP


UIKit independent mediator
UIView and/or UIViewController
Presenter
Owns and
sends user actions
Owns and updates
View
(passive)
Updates
Notifies
Model

MVP的评价

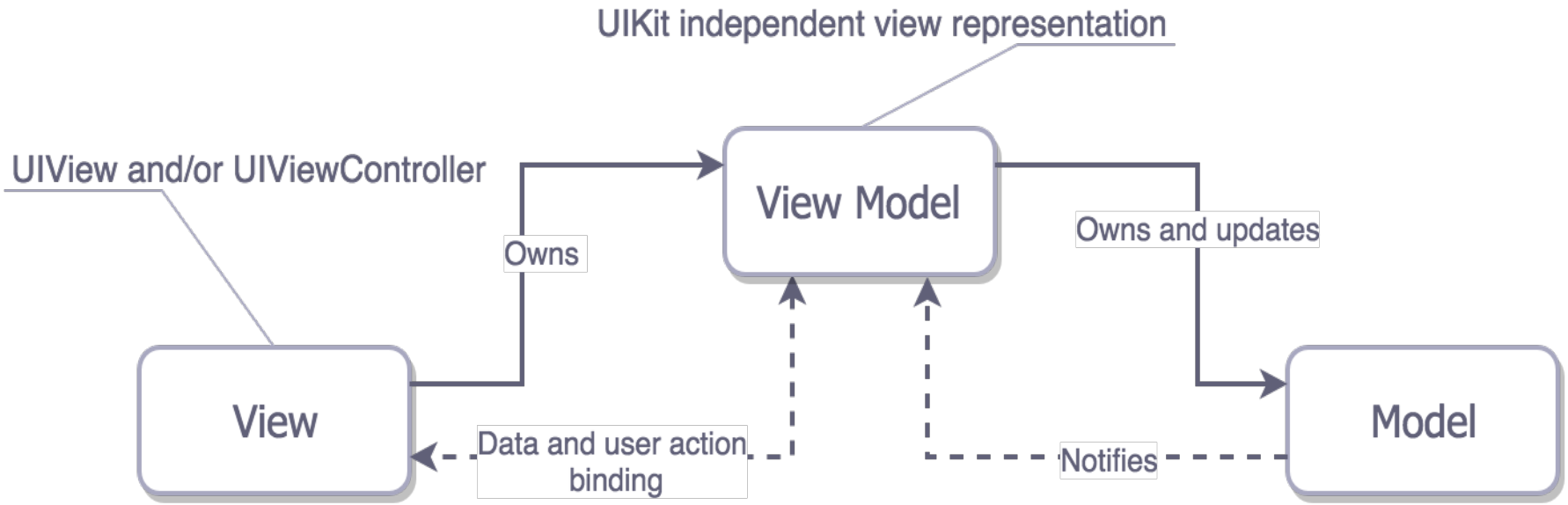


- 职责划分较好

- 可测性较强

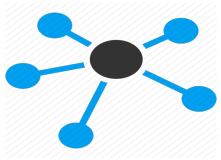
- 代码量增大

Microsoft MVVM



数据绑定

MVVM的评价

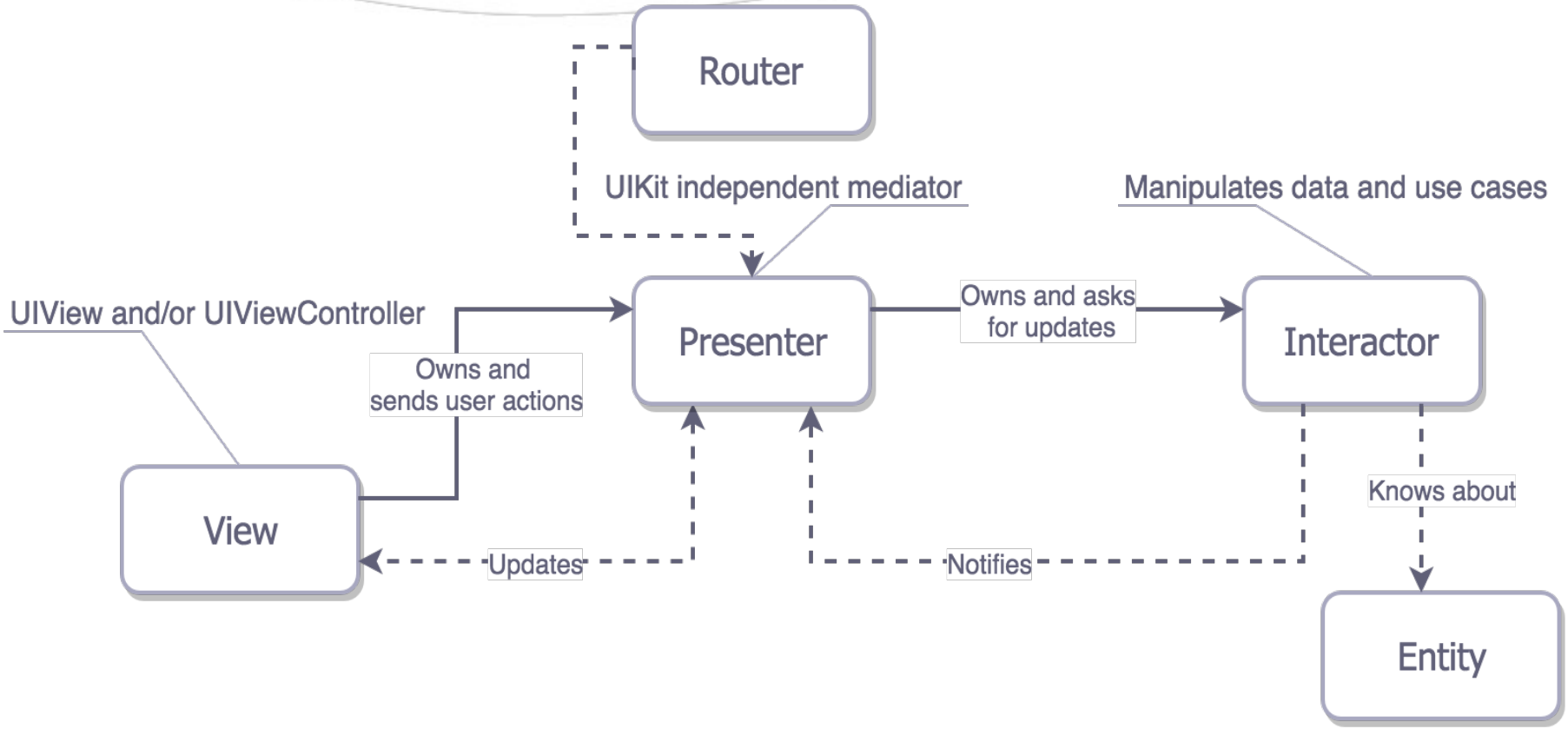


- 职责划分较好

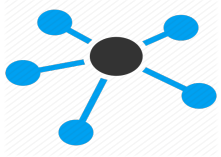
- 可测性较强

- 代码量增大，响应式编程门槛

VIPER


Router
UIKit independent mediator
Manipulates data and use cases
UIView and/or UIViewController
Presenter
Owns and asks for updates
Interactor
Owns and sends user actions
View
Knows about
Updates
Notifies
Entity

VIPER的评价

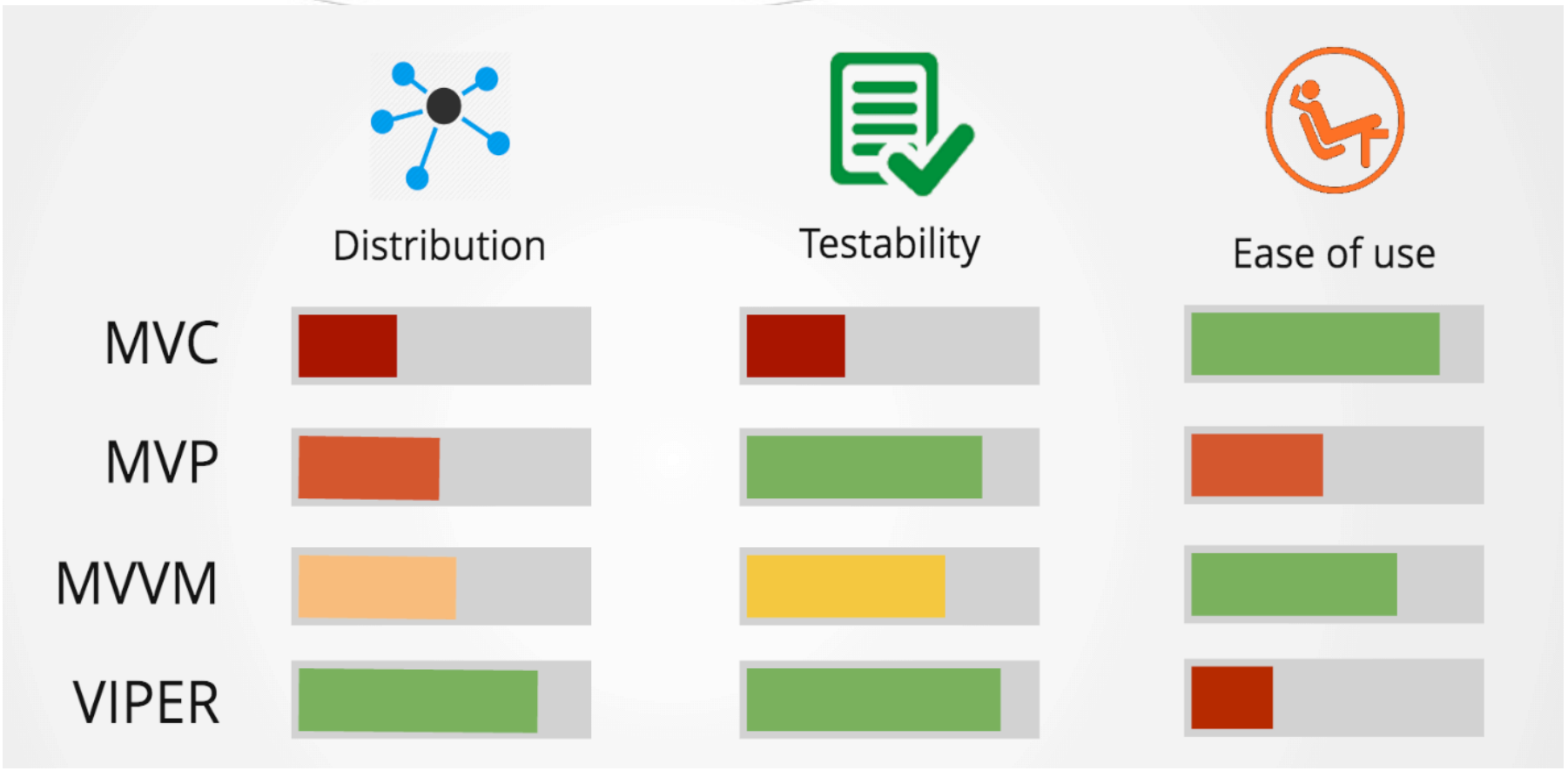


- 职责划分好

- 可测性强

- 代码量庞大，基于用例的应用设计

综合比较



总结

我们已经从头到尾地了解了几种架构模式，希望你能从中找到那些曾经困扰你很久的问题的答案。但我毫不怀疑，你已经意识到了没有什么银色子弹，选择什么样的架构设计是特定场景下权衡各种因素之后的结果。因此，在同一个app中就会出现**混合架构设计**。比如：一开始使用**MVC**，然后你发现有一些特殊场景如果使用**MVC**将会难以维护，这时你可以仅对这个场景使用**MVVM**模式，没必要去重构那些**MVC**架构执行的很好的模块。**MV(X)**系列是互相兼容的。

Make everything as simple as possible, but not simpler.

-- Albert Einstein