<https://habr.com/ru/company/badoo/blog/281162/>

https://www.youtube.com/watch?v=2z5uWBRGjFI&t=416s

ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

Зачем нужна архитектура:

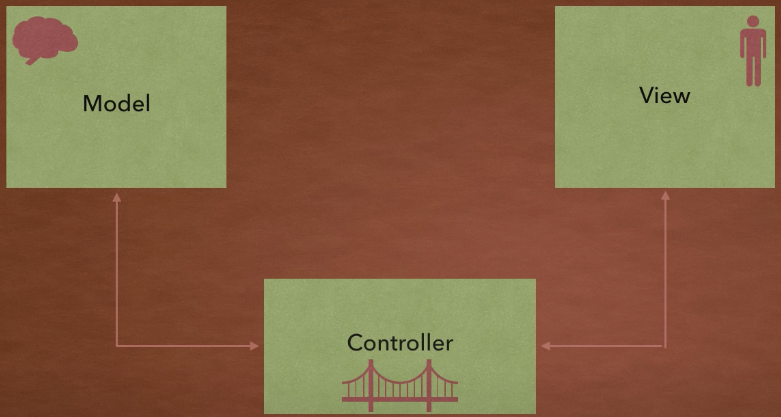
* Архитектура - это фундамент благодаря которому можно достичь единообразного и понятного кода во всем приложении;
* Делает код поддерживаемым, удобным для модификации и добавления нового функционала;
* Позволяет вести параллельную разработку модулей.

Признаки хорошей архитектуры:

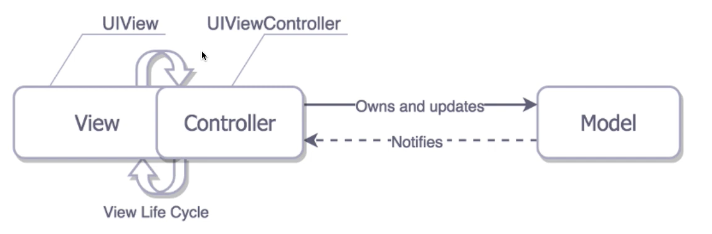
* Сбалансированное распределение обязанностей между сущностями с жесткими ролями;
* Тестируемость;
* Простота использования и низкая стоимость обслуживания.

MVC от Apple

Хорошо используется если проект не сильно большой.



В реальности выглядит так: Massive View Controller



View и Controller тесно связаны

* View – это окно через которое пользователь взаимодействует с приложением. Чаще всего вид отображает содержимое модели, но, кроме того он же воспринимает действия пользователя. Любые контакты между пользователем и приложением отправляются в вид. После этого они могут быть перехвачены Controller и переданы в Model;
* Model – это мозги приложения. Она выполняет все вычисления и создает для себя виртуальный мир, в котором может существовать сама без вычислений видов и контролеров. Иными словами можно считать модель виртуальной копией приложения без интерфейса;
* Controller – это контроллеры видов (ViewControllers). Контроллер является переходным звеном между model и view. Он интерпретирует события происходящие с одной стороны и по мере необходимости использует эту информацию для внесения изменений на другой стороне. Например: если пользователь изменяет какое-либо поле в виде то контроллер гарантирует что и модель измениться соответствующим образом. А если модель получит новые данные (например из интернета), то контроллер прикажет виду отобразить их.

Переходы между модулями делаются через слой View. Потому что другого слоя для этого нет. Model отвечает за бизнес логику а переходы это не бизнес логика. Presenter за это отвечать тоже не должен, остается слой View.

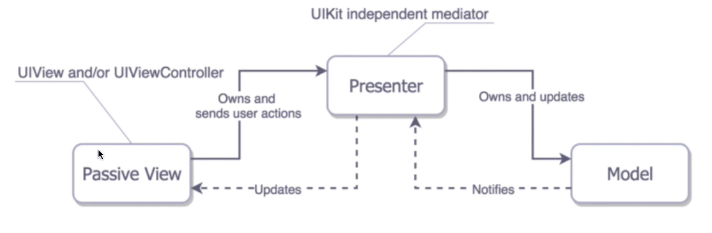
Минусы паттерна:

* Легко превращается в Massive View Controller;
* View и Controller тесно связаны;
* Тяжело тестировать Unit тестами VC.

Плюсы паттерна:

* Позволяет быстро разрабатывать;
* Низкий порог входа новичка.

MVP в идеале так должен был работать MVC



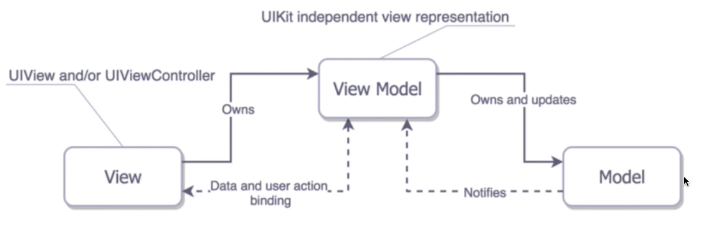
* Model –
* Passive View –
* Presenter – столько же как и экранов.

Плюсы и минусы:

* Ответственность распределена между M&P;
* View слабо связано и пассивно;
* Высококачественное Unit тестирование.
* Кол-во кода в 2 раза больше чем в MVC;
* Немного увеличивается время разработки .

Переходы между модулями делаются через слой View. Потому что другого слоя для этого нет. Model отвечает за бизнес логику а переходы это не бизнес логика. Presenter за это отвечать тоже не должен, остается слой View.

MVVM – Реактивное программирование



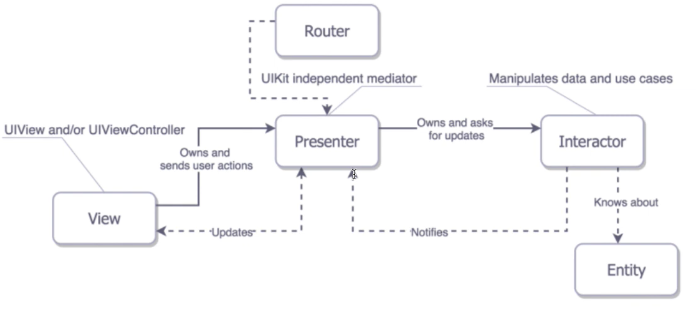
* Model – какой-то слой в приложении который работает с бизнес логикой. Этот слой общий, не обязательно тут бизнес логика одного экрана;
* View Model – создается каждый раз для одного экрана. Является классом который содержит минимальную достаточную информацию чтобы отобразить данные во View;
* View – создается каждый раз для одного экрана. Отвечает за отображение.

Плюсы и минусы:

* Логика вынесена в ModelView, VC чиста;
* View слабо связано и пассивно;
* Высококачественное Unit тестирование;
* Выглядит лучше чем MVP;
* Есть библиотеки для биндингов (RxSwift).
* Кол-во кода в 2 раза больше;
* Немного увеличивается время разработки;
* Чем больше сила тем больше ответственности.

Переходы между модулями делаются через слой View. Потому что другого слоя для этого нет. Model отвечает за бизнес логику а переходы это не бизнес логика. Presenter за это отвечать тоже не должен, остается слой View.

VIPER – Наибольшее деление на слои



* Router – слой отвечающий за передачу данных между модулями и переходами между ними;
* Presenter – слой для общения со всеми слоями;
* View – отображение;
* Interactor – слой содержащий в себе всю бизнес логику. Т.е это Model из прошлых архитектур;
* Entity – маленькая модель которая нужна только одному нашему модулю, а не всему проекту;
* Assembly – слой в котором идет сборка модуля.

Отличие Viper от MV(x) паттернов:

* Логика из Model (взаимодействие данных) смещается в Interactor, а так же есть Entities – структуры данных, которые ничего не делают;
* Из Controller, Presenter , ViewModel обязанности представления UI переехали в Presenter, но без возможности изменения данных;
* Viper является первым шаблоном, который пробует решить проблему навигации, для этого есть Router.

Плюсы и минусы:

* Еще детальнее распределены отношения;
* Каждый модуль можно пере использовать;
* Удовлетворяет принципам SOLID;
* Высококачественное Unit тестирование.
* Кол-во кода в 5 раз больше;
* Сильно увеличивается время разработки.