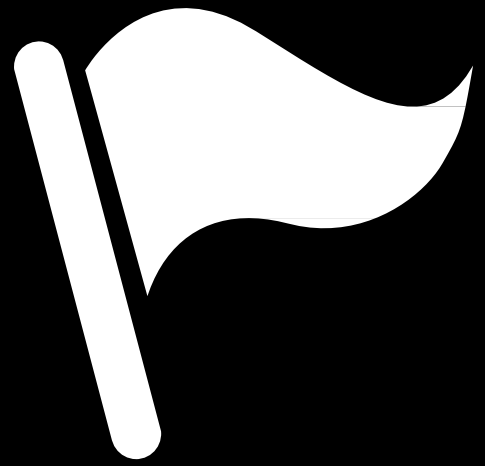
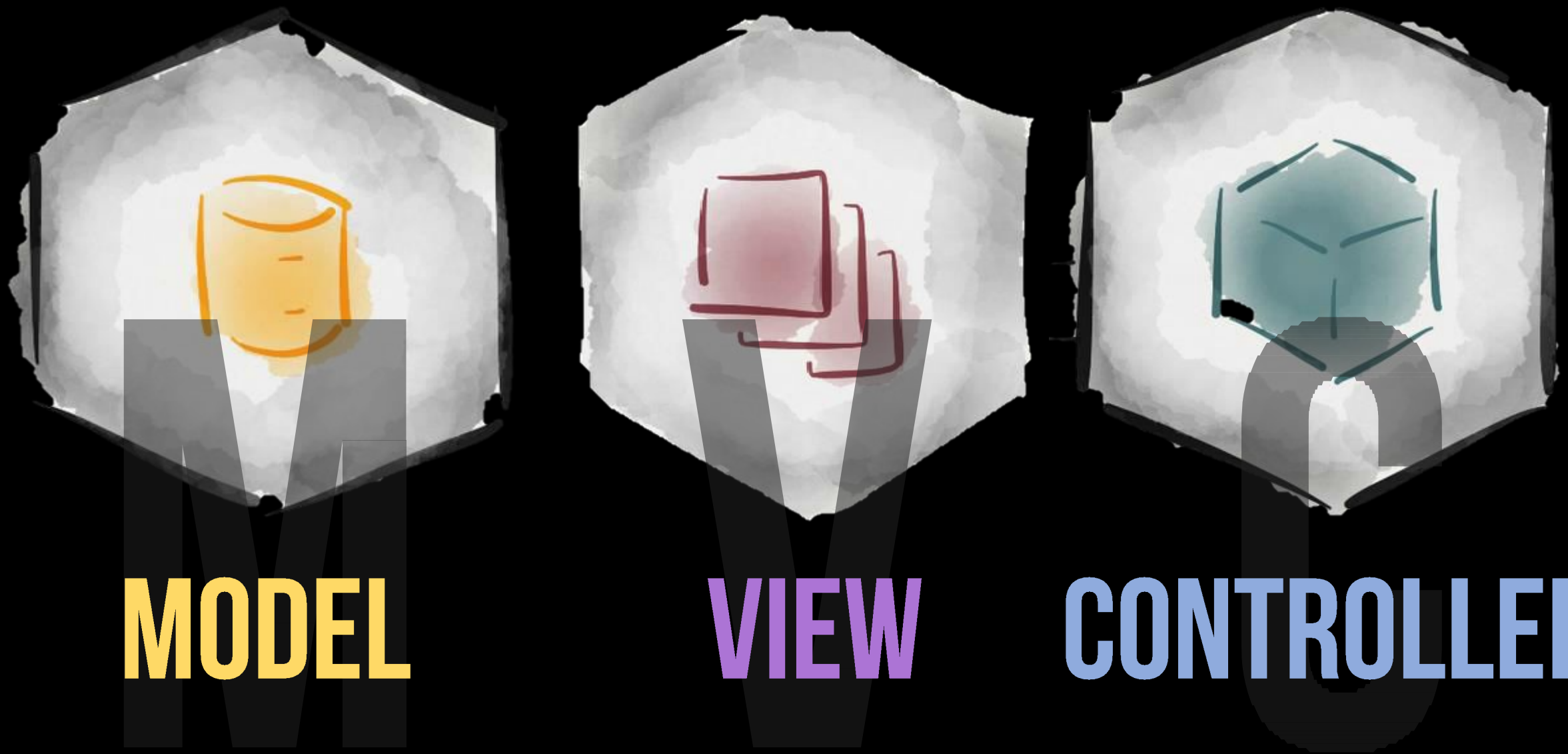


Федоренко
Никита

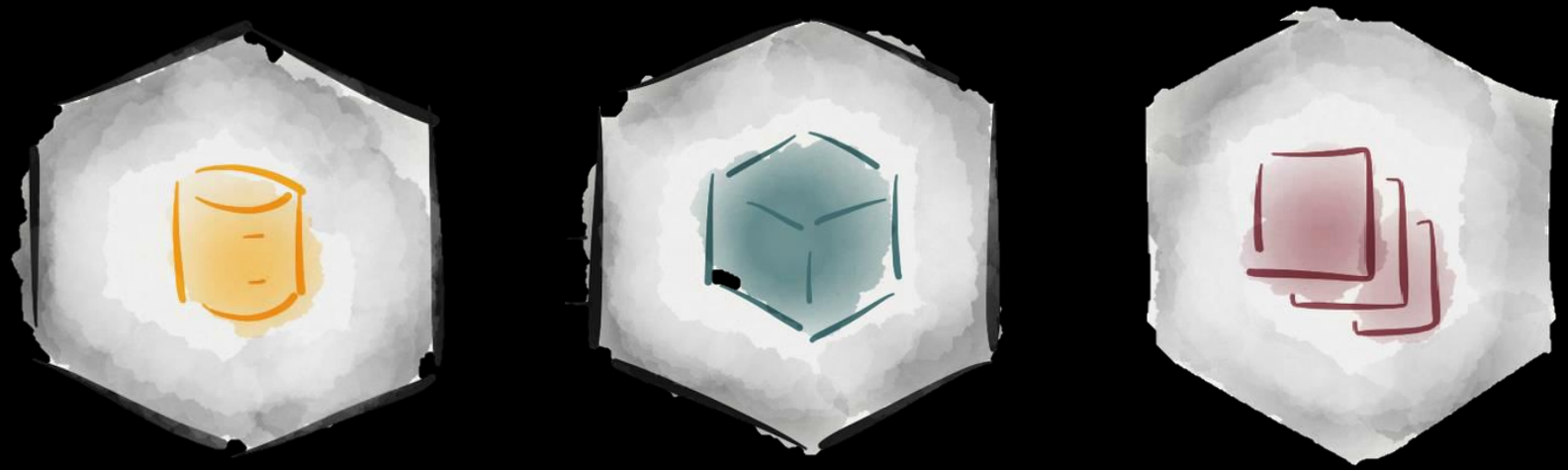
iOS Developer

MVC


MODEL
VIEW
CONTROLLER

MODEL VIEW CONTROLLER (MVC)

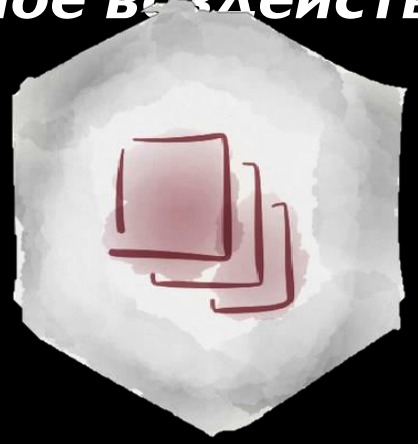
- MVC предназначен для разделения бизнес-логики и пользовательского интерфейса.



MVC PATTERN

- ***Паттерн проектирования с помощью которого модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на оста***

One Model



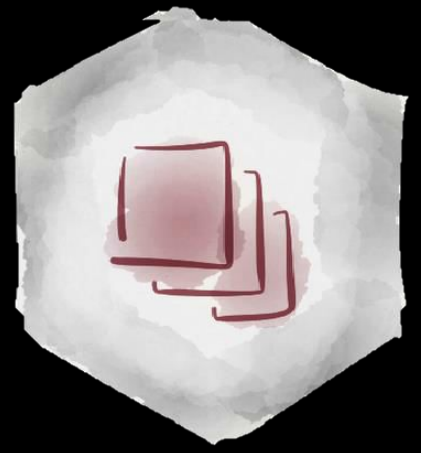
Many Views

Many Controllers

MVC PATTERN

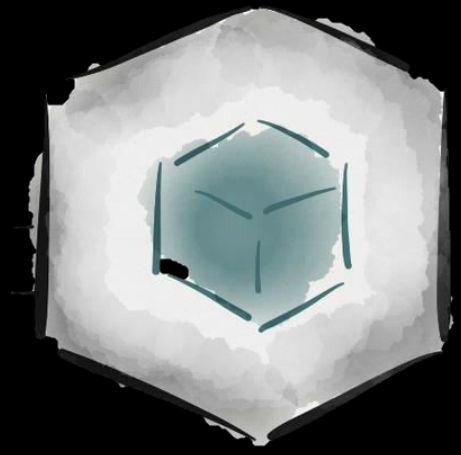
Model

Бизнесс логика
Хранилище данных



View

Элементы
Интерфейса



Controller

Посредник между моделью иView
Обрабатывает действия
пользователя и изменения модели

MODEL

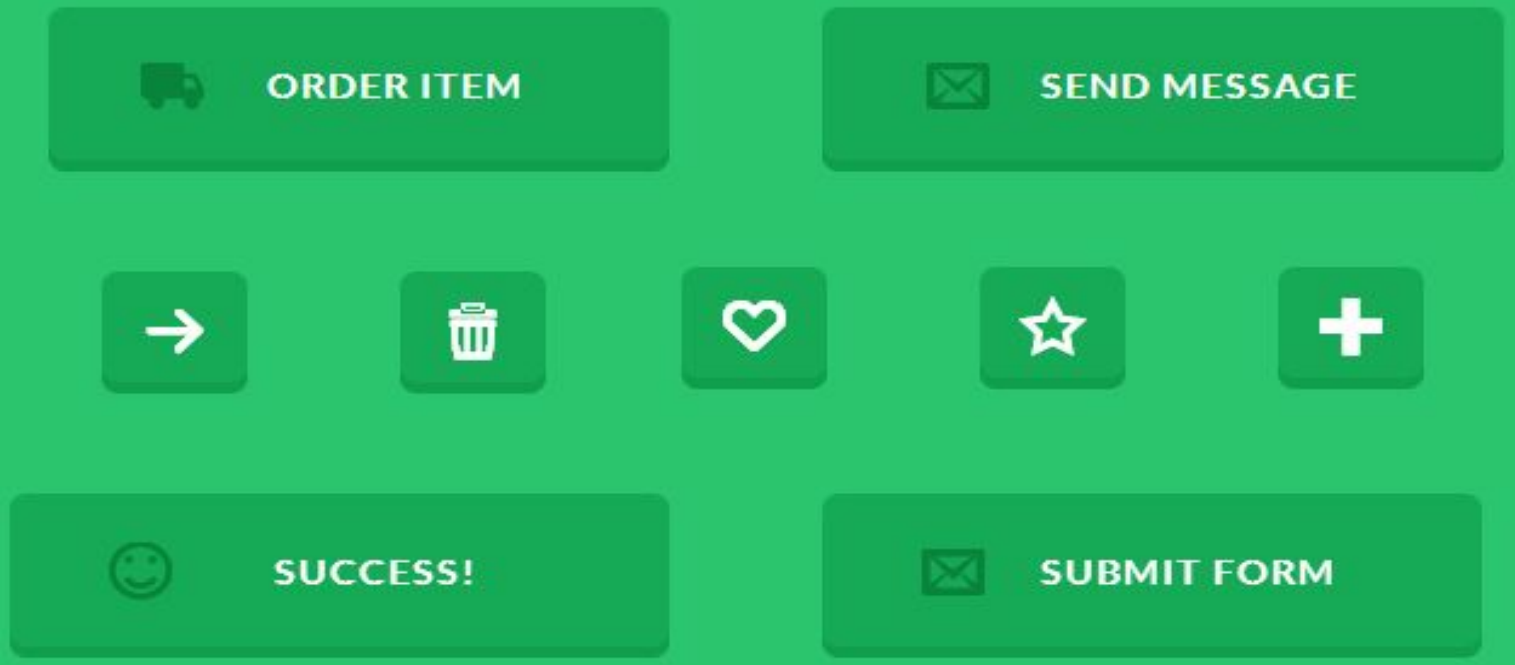
Под Моделью, обычно понимается часть содержащая в себе функциональную бизнес-логику приложения. Модель должна быть полностью независима от остальных частей продукта. Модельный слой ничего не должен знать об элементах дизайна, и каким образом он будет отображаться. Достигается результат, позволяющий менять представление данных, то как они отображаются, не трогая саму Модель

VIEW

В обязанности Представления входит отображение данных полученных от Модели. Однако, представление не может напрямую влиять на модель. Можно говорить, что представление обладает доступом «только на чтение» к данным.

CONTROLLER

Под контроллером обычно понимают посредник между Model и View; в целом отвечает за изменения Model, реагируя на действия пользователя, выполненные на View, и обновляет View, используя изменения из Model



HISTORY

- Концепция MVC была описана Трюгве Реенскаугом в 1979 году, работавшим в то время над языком программирования «SmallTalk» в научно-исследовательском центре «Xerox PARC».

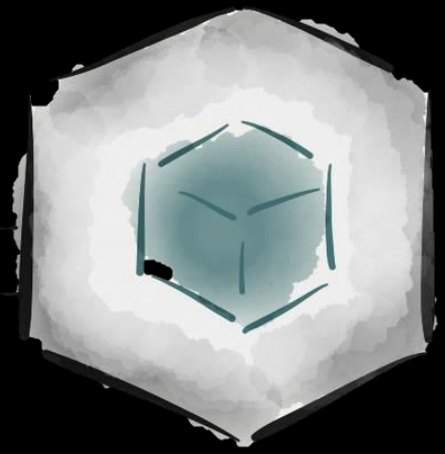
- Различают две основные модификации : Активная модель и Пассивная модель.

THE MVC PATTERN

Control Flow

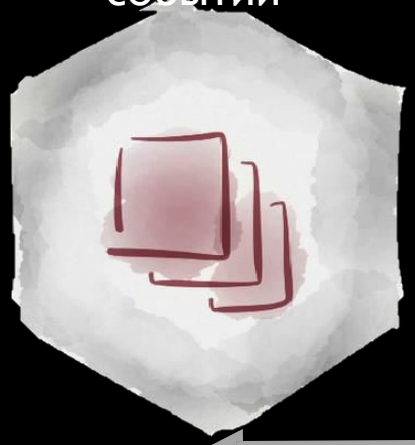
1. Пользователь взаимодействует с интерфейсом.
2. Контроллер получает событие.
3. Контроллер сообщает модели о действиях пользователя, что может привести к изменению модели.
4. View берет данные из модели и на основании их изменяется.
5. Интерфейс (View) ждет последующих

THE MVC PATTERN



3 Контроллер меняет модель

2 View сообщает контроллеру о событии



5 View берет данные из модели и меняется

4 Model сообщает View что данные изменились

1 Пользователь взаимодействует с View

BENEFITS

- Организация
- Быстрая разработка приложения
- Переиспользование кода
- Параллельная разработка
- Изображение одой информации разными способами
- Быстрая реакция на изменение данных.

MVC (Проблемы)

1. Отделение контроллера от вида
2. Отделение модели(M) от вида(V).
3. Отделение модели от контроллера(C)
4. Иерархия вложенных MVC
5. Создание множества лишних классов.
6. Проблема вызовов (Производительность)
7. Где хранить промежуточные данные? в контроллере?

MVP

Model View Presenter (MVP)

- Model-View-Presenter (MVP) — шаблон проектирования, производный от MVC, который используется в основном для построения пользовательского интерфейса.
- **Элемент Presenter в данном шаблоне берёт на себя функциональность посредника (аналогично контроллеру в MVC) и отвечает за управление событиями пользовательского интерфейса (например, использование мыши) так же, как в других шаблонах обычно отвечает представление.**

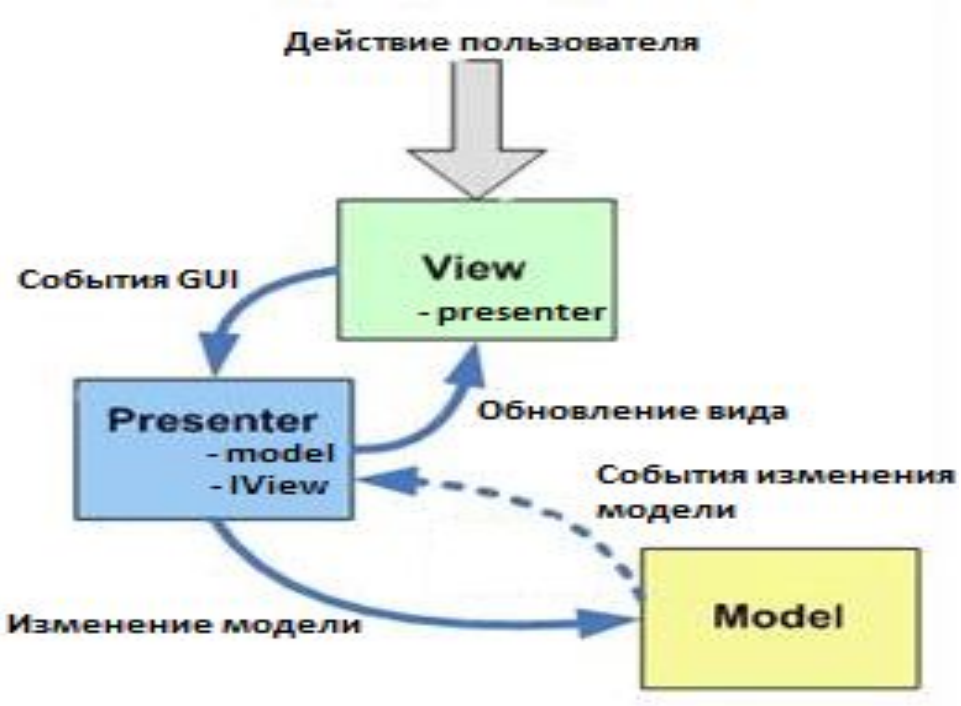
MVP

- Шаблон MVP позволяет отделить уровень представления от уровня логики, для того что бы поведение приложения не зависело от его конкретного внешнего вида. В идеале, используя MVP мы добьемся того, чтобы одна и та же логика могла иметь совершенно разные, а главное взаимозаменяемые UI представления.

Первое, что нужно понять это то, что MVP не архитектурный шаблон, т.е он ответственен только за уровень представления. Тем не менее, его использование улучшает вашу архитектуру.

Model-view-presenter

- Контроллер (Presenter) дает знать представлению об изменениях.
- Данный подход позволяет создавать абстракцию представления



Model-View-Presenter

Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.

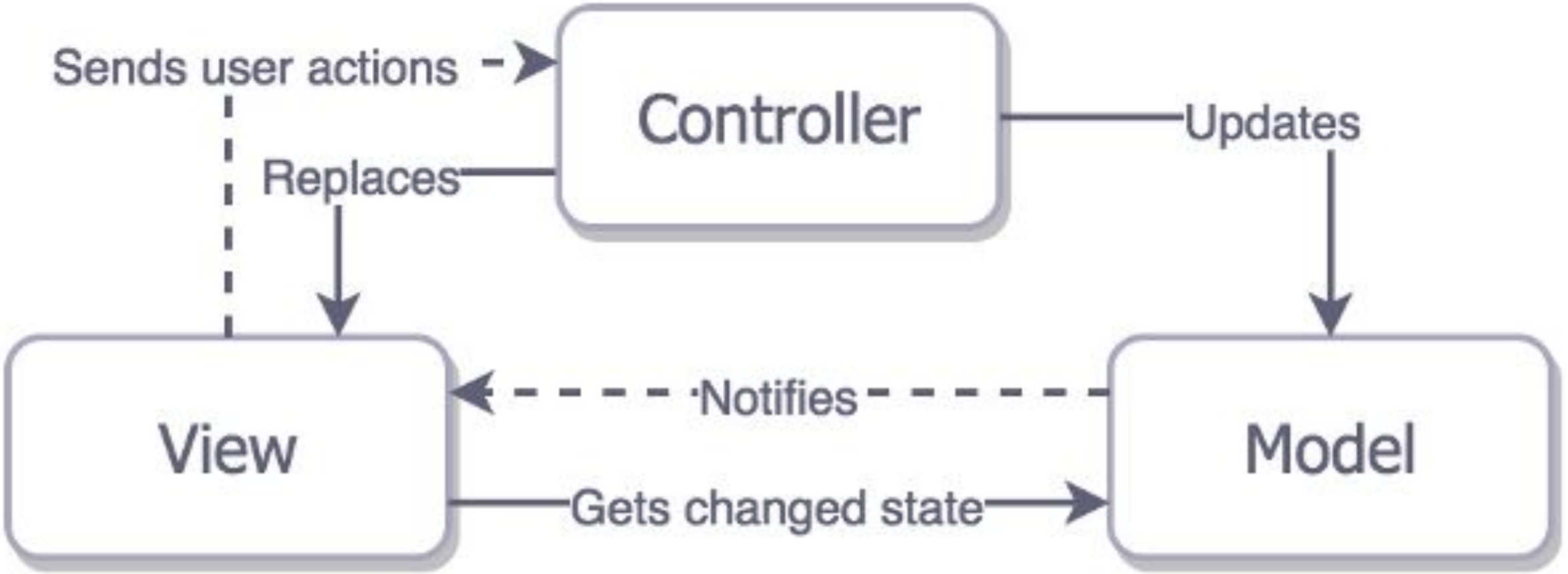
Признаки презентера:

- Двухсторонняя коммуникация с представлением;
- Представление взаимодействует напрямую с презентером, путем вызова соответствующих функций или событий экземпляра презентера;
- Презентер взаимодействует с View путем использования специального интерфейса, реализованного представлением;
- Один экземпляр презентера связан с одним отображением

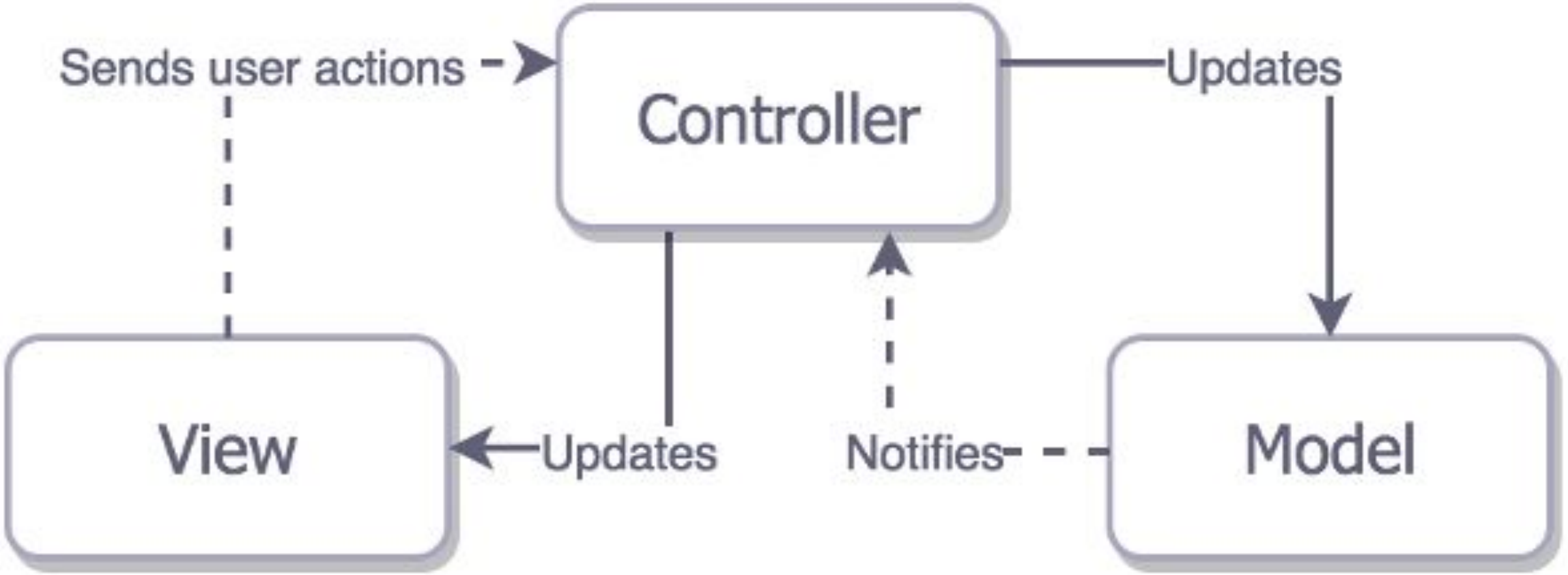
Отличия от MVC

- Более слабая связь с моделью. Presenter отвечает за связывание модели и представления
- Более легкая организация тестирования, т.к. взаимодействие с представлением идет через интерфейс

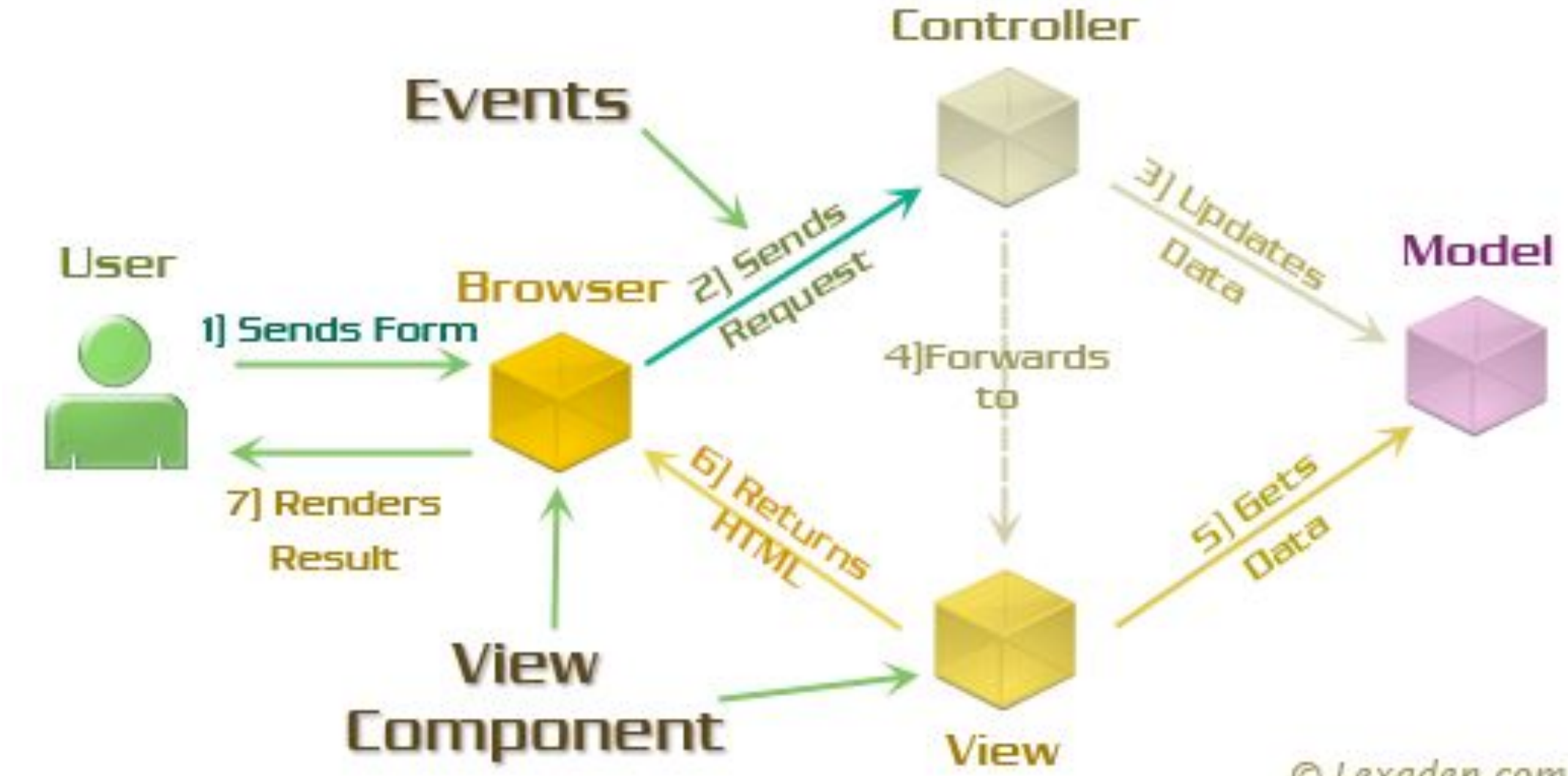
MVC


Sends user actions
Controller
Updates
Replaces
View
Notifies
Model
Gets changed state

MVP


Sends user actions
Controller
Updates
View
Updates
Notifies
Model

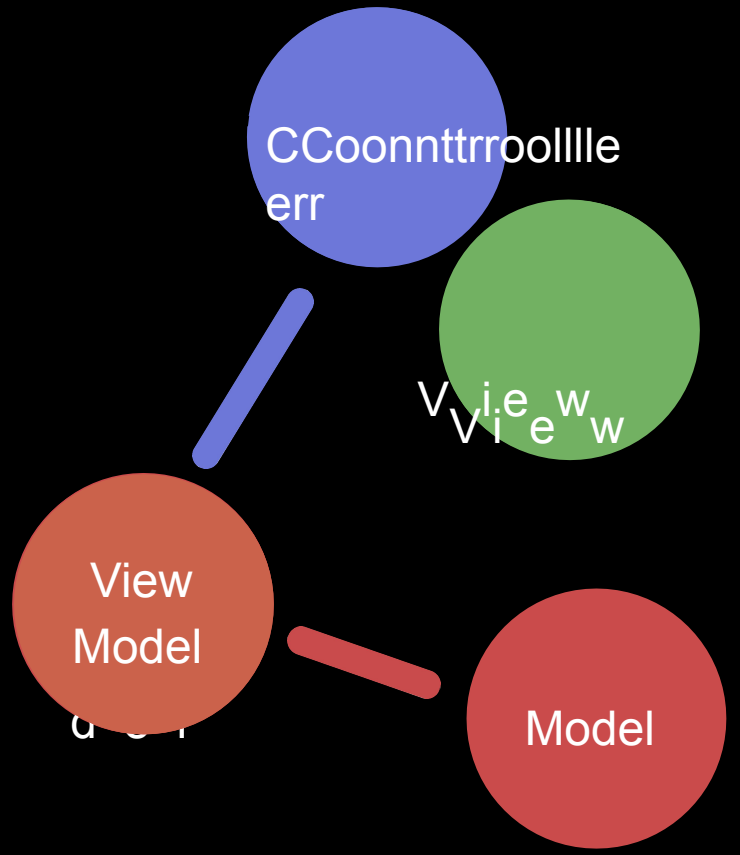
Model View Controller



Model View Presenter

MVVM

MVVM



MVVM

Признаки View-модели:

- Двухсторонняя коммуникация с представлением;
- View-модель — это абстракция представления. Обычно означает, что свойства представления совпадают со свойствами View-модели / модели
- View-модель не имеет ссылки на интерфейс представления (IView). Изменение состояния View-модели автоматически изменяет представление и наоборот, поскольку используется механизм связывания данных (Bindings)
- Один экземпляр View-модели связан с одним отображением.

MVVM

Реализация:

При использовании этого паттерна, представление не реализует соответствующий интерфейс (IView).

Представление должно иметь ссылку на источник данных (DataContex), которым в данном случае является View-модель. Элементы представления связаны (Bind) с соответствующими свойствами и событиями View-модели.

В свою очередь, View-модель реализует специальный интерфейс, который используется для автоматического обновления элементов представления.

MV
C

```
struct    Person
   var{ firstName:    String
   var  secondName   String
}       :
```

MV
C

```
struct Person {
    var firstName: String
    var secondName: String
}

class ViewController: UIViewController {
    var person: Person!
    @IBOutlet var nameLabel: UILabel!
}
```

MV
C

```
struct    Person
   var{ firstName:    String
   var  secondName   String
}       :

class   ViewController:     UIViewController  {
   var  person:  Person!
   @IBOutlet  var  nameLabel:   UILabel!

   override  func viewDidLoad()     {
      super.viewDidLoad()
      nameLabel.text    =    "\(person.firstName)  \(person.secondName)"
   }
}
```

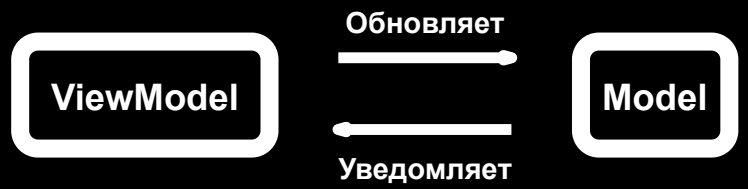
MVVM

```
struct    Person
   va{ firstName:    String
   var  secondName   String
}       :

struct   PersonModel  {
   var  person:   Person
   var  name:    String
      return {"\(person.firstName)      \(person.secondName)"
   }
}

class    ViewController:      UIViewController      {
    va   person:  Person!
    r

   override  func viewDidLoad()      {
      super.viewDidLoad()
      nameLabel.text    =    "\(person.firstName)  \(person.secondName)"
   }
}
```

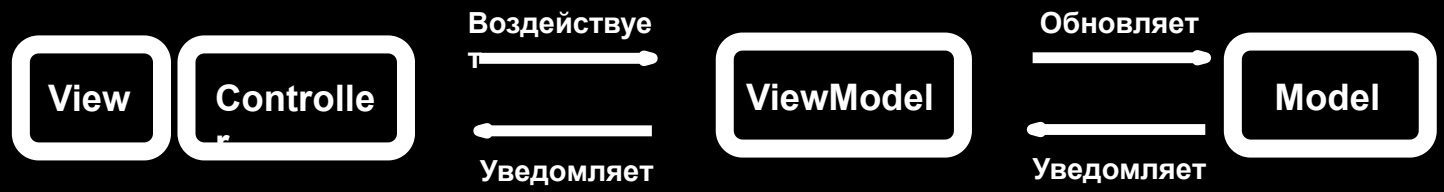
MVVM

```
struct    Person
   va{ firstName:    String
   var  secondName   String
}       :

struct   PersonModel  {
   var  person:   Person
   var  name:    String
      return {"\(person.firstName)     \(person.secondName)"
   }
}

class   ViewController:     UIViewController  {
   var  personModel:   PersonModel!
   @IBOutlet  var  nameLabel:   UILabel!

   override     func viewDidLoad()      {
      super.viewDidLoad() nameLabel.text=
            personModel.name
   }
}
```

MVVM
Схема



MVVM
Схема


View
Controller
Воздействует
Уведомляет
ViewModel
Обновляет
Уведомляет
Model

MVVM
Схема

- Обновляет:
  - gameSession.attempts += 1
- Воздействует:
  - gameSessionEntity.loseAction()