**СРАВНЕНИЕ MVP И MVVM**

**Model-View-Presenter (MVP)**

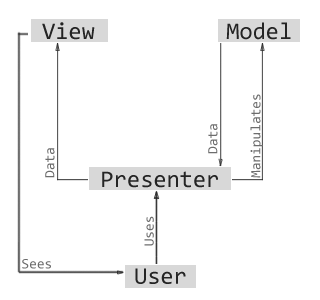
**Model-View-Presenter (MVP)** — [шаблон проектирования](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), производный от [MVC](https://ru.wikipedia.org/wiki/Model-View-Controller), который используется в основном для построения [пользовательского интерфейса](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D0%BA%D0%B8%D0%B9_%D0%B8%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81).

**MVP** — шаблон проектирования пользовательского интерфейса, который был разработан для облегчения автоматического [модульного тестирования](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) и улучшения [разделения ответственности](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D0%B7%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D0%B8) в презентационной логике (отделения логики от отображения):

*Модель* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Model*) — данные для отображения;

*Вид* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *View*) — реализует *отображение* данных (из Модели), обращается к Presenter за обновлениями, перенаправляет события от пользователя в Presenter;

*Представитель* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *Presenter*) — реализует взаимодействие между Моделью и Видом и содержит **в себе всю логику представления данных о предметной области;** при необходимости получает данные из хранилища и преобразует для отображения во View.



Обычно экземпляр Вида (Представление) создаёт экземпляр Представителя, передавая ему ссылку на себя. При этом Представитель работает с Видом в абстрактном виде, через его [интерфейс](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). Когда вызывается событие Представления, оно вызывает конкретный метод Представителя, не имеющего ни параметров, ни возвращаемого значения. Представитель получает необходимые для работы метода данные о состоянии пользовательского интерфейса через интерфейс Вида и через него же передаёт в Вид данные из Модели и другие результаты своей работы.

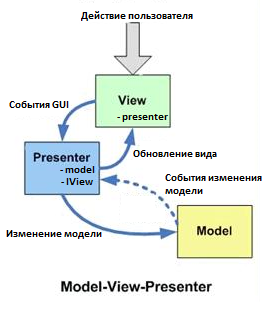
Данный подход позволяет создавать абстракцию представления. Для этого необходимо выделить интерфейс представления с определенным набором свойств и методов. Презентер, в свою очередь, получает ссылку на реализацию интерфейса, подписывается на события представления и по запросу изменяет модель.  
  
**Признаки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

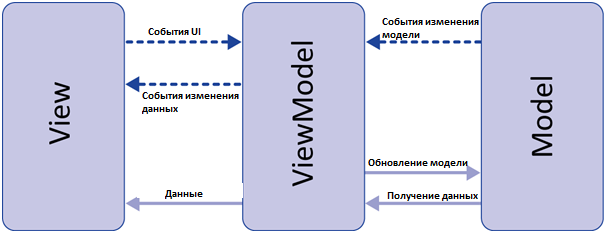


**Реализация:**  
Каждое представление должно реализовывать соответствующий интерфейс. Интерфейс представления определяет набор функций и событий, необходимых для взаимодействия с пользователем (например, **IView**.ShowErrorMessage(string msg)). Презентер должен иметь ссылку на реализацию соответствующего интерфейса, которую обычно передают в конструкторе.  
Логика представления должна иметь ссылку на экземпляр презентера. Все события представления передаются для обработки в презентер и практически никогда не обрабатываются логикой представления (в т.ч. создания других представлений).

**Model-View-View Model**

**Model-View-ViewModel** (MVVM) — шаблон проектирования архитектуры приложения. Представлен в 2005 году Джоном Госсманом (John Gossman) как модификация шаблона Presentation Model.

Используется для разделения модели и её представления, что необходимо для их изменения отдельно друг от друга. Например, разработчик задаёт логику работы с данными, а дизайнер работает с пользовательским интерфейсом.



Данный подход позволяет связывать элементы представления со свойствами и событиями View-модели. Можно утверждать, что каждый слой этого паттерна не знает о существовании другого слоя.  
  
**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.*

**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. Примером такого интерфейса в WPF может быть INotifyPropertyChanged.

Общие правила выбора паттерна

**MVC**

Используется в ситуации, когда связь между представление и другими частями приложения невозможна (и Вы не можете использовать MVVM или MVP);

Частым примером использования может служить ASP.NET MVC.

**MVVM**

Используется в ситуации, когда возможно связывание данных без необходимости ввода специальных интерфейсов представления (т.е. отсутствует необходимость реализовывать IView);

Частым примером является технология WPF.

**Выводы**

В заключении хочется отметить, что строго придерживаться только одному паттерну — не всегда лучший выбор. В зависимости от ситуации нужно отделить представление от бизнес логики и логики, которая их связывает.