PRACTICAL MVVM

@ksmandersen

PRACTICAL MVVM

The Model View ViewModel (MVVM) is an architectural pattern used in software engineering that originated from Microsoft as a specialization of the Presentation Model design pattern introduced by Martin Fowler

The Model View ViewModel (MVVM) is an architectural pattern used in software engineering that originated from Microsoft as a specialization of the Presentation Model design pattern introduced by Martin Fowler

Microsoft


Windows
An error has occurred. To continue:
Press Enter to return to Windows, or
Press CTRL+ALT+DEL to restart your computer. If you do this,
you will lose any unsaved information in all open applications.
Error: 0E : 016F : BFF9B3D4
Press any key to continue _

Largely based on the model–view–controller pattern (MVC), MVVM is a specific implementation targeted at UI development platforms which support event-driven programming, specifically Windows Presentation Foundation (WPF) and Silverlight on the .NET platforms using XAML and .NET languages. Technically different, but similar, Presentation Model design patterns are available in HTML5 through AngularJS, KnockoutJS, Ext JS, Vue.js, and for Java the ZK framework (Model-View-Binder).

Largely based on the model–view–controller pattern (MVC), MVVM is a specific implementation targeted at UI development platforms which support event-driven programming, specifically Windows Presentation Foundation (WPF) and Silverlight on the .NET platforms using XAML and .NET languages. Technically different, but similar, Presentation Model design patterns are available in HTML5 through AngularJS, KnockoutJS, Ext JS, Vue.js, and for Java the ZK framework (Model-View-Binder).

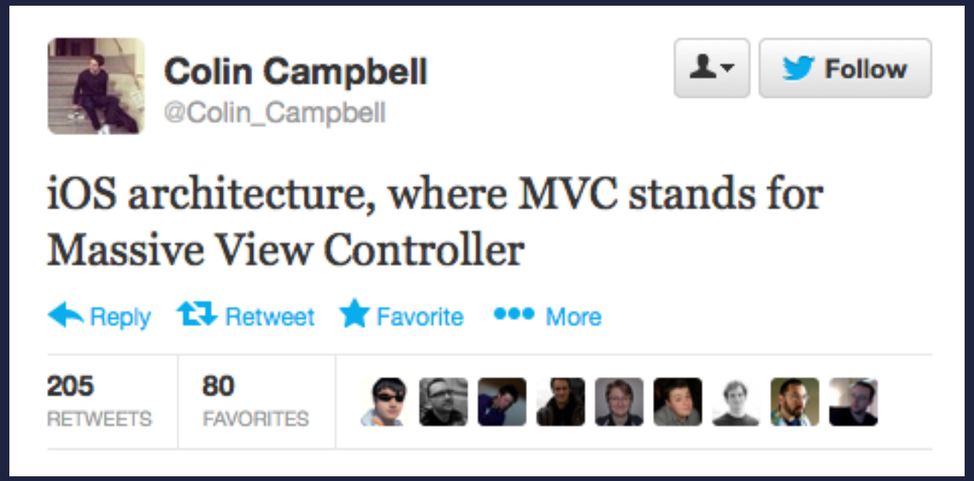
MVVM

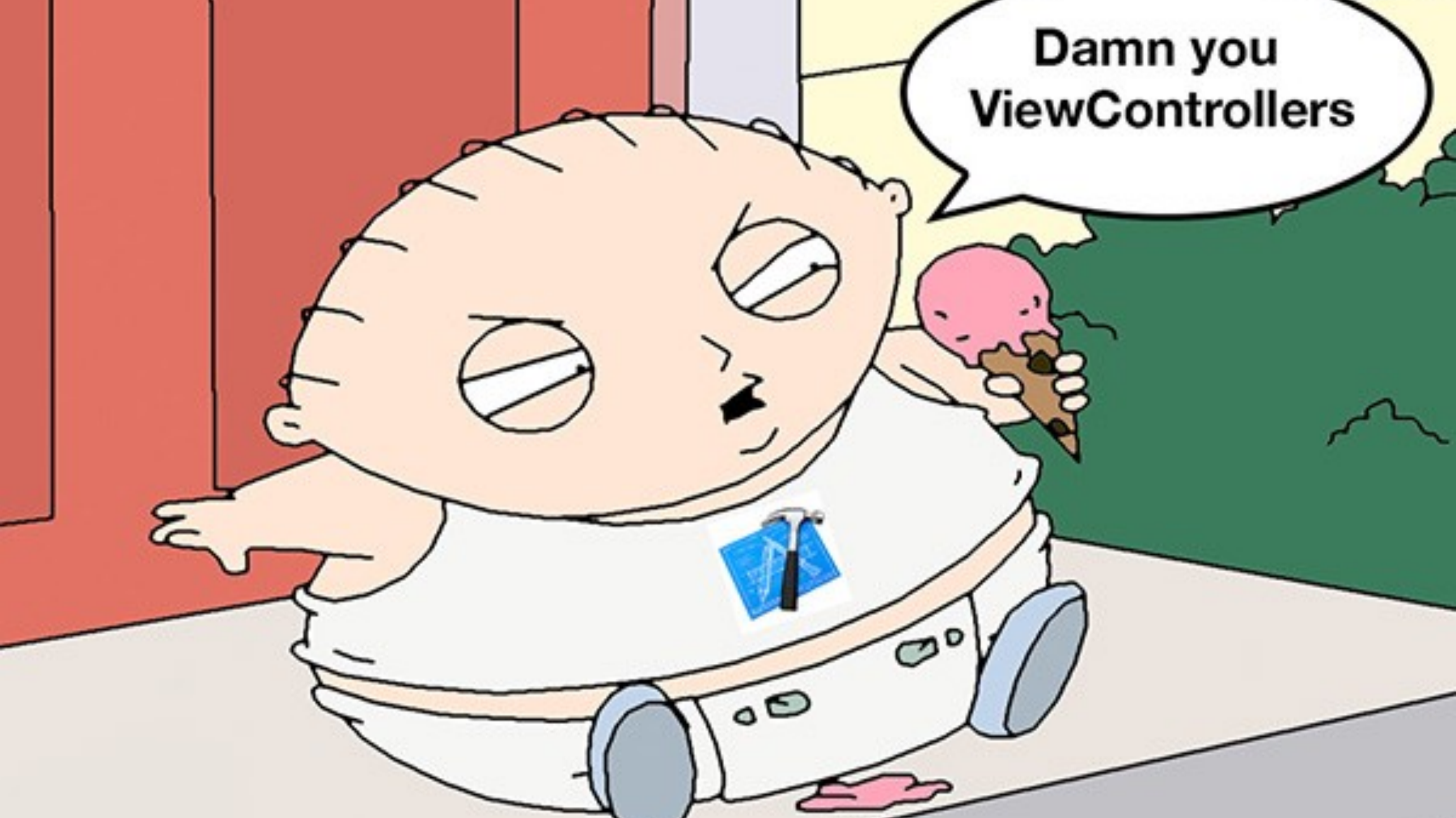
WITHOUT REACTIVECOCOA

MVVM

WITHOUT ~~REACTIVECOCOA~~

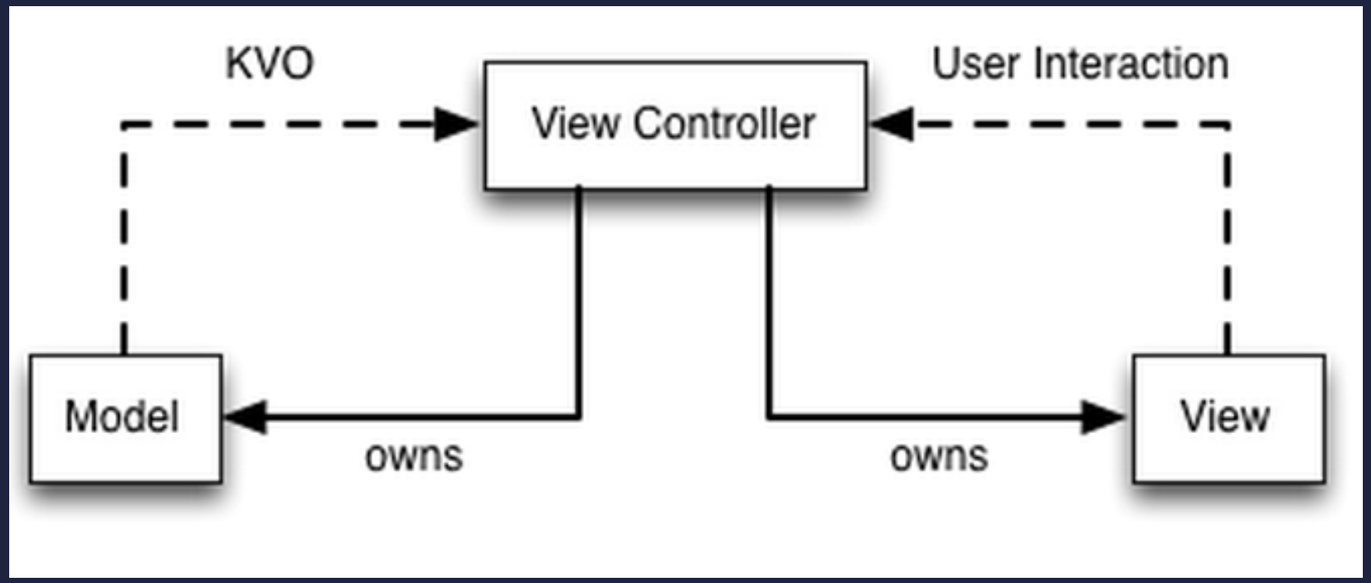
MVC

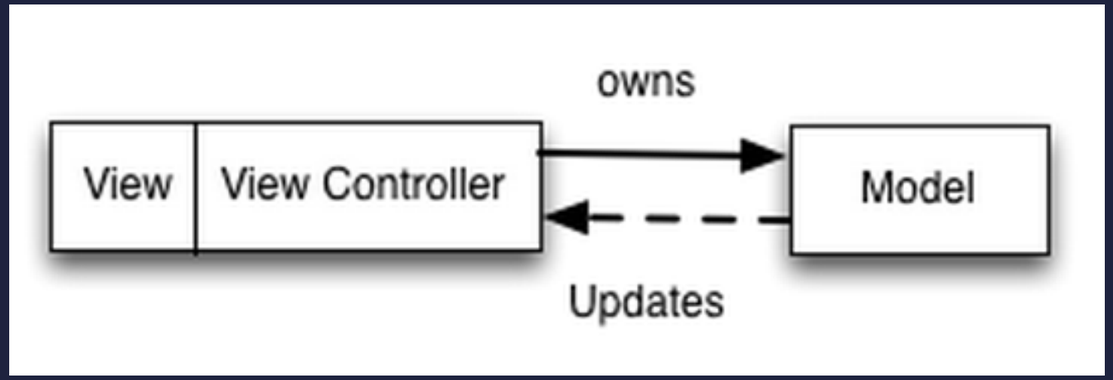

Colin Campbell
@Colin_Campbell
Follow
iOS architecture, where MVC stands for Massive View Controller
Reply
Retweet
Favorite
More
205
RETWEETS
80
FAVORITES

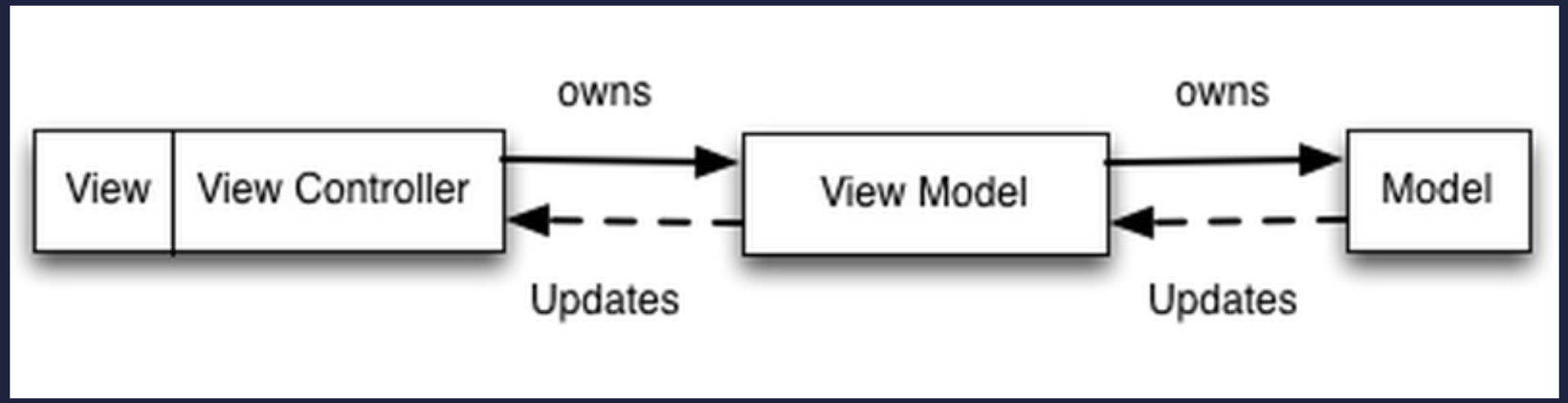

Damn you
ViewControllers

MASSIVE VIEWCONTROLLER

MISTREATED VIEWCONTROLLERR


KVO
View Controller
User Interaction
Model
owns
owns
View


owns
View
View Controller
Model
Updates


owns
owns
View
View Controller
View Model
Model
Updates
Updates

```
class Show: RLMObject {
    dynamic var identifier: Int
    dynamic var name: String
    dynamic var posterPath: String
    dyanmic var firstAired: NSDate
}
```

```
class ShowListViewController: UIViewController, UICollectionViewDataSource {
    func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int {
        return Int(fetchedResultsController.sections.count)
    }

    func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        if let info = fetchedResultsController.sections[section] {
            return info.numberOfObjects
        }

        return 0
    }

    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier, forIndexPath: indexPath) as ShowCell
        if let show = fetchedResultsController.objectAtIndexPath(indexPath) {
            cell.titleLabel.text = cell.name
            if let posterURL = NSURL(string: show.posterPath) {
                cell.posterImageView.hnk_setImageFromURL(posterURL)
            }

            if let firstAired = someFormatter.dateFromString(show.firstAired) {
                cell.firstAiredLabel.text = firstAired
            }
        }

        return cell
    }
}
```

```swift
class ShowListViewController: UIViewController, UICollectionViewDataSource {
    func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int {
        return Int(fetchedResultsController.sections.count)
    }

    func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        if let info = fetchedResultsController.sections[section] {
            return info.numberOfObjects
        }

        return 0
    }

    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier, forIndexPath: indexPath) as ShowCell
        if let show = fetchedResultsController.objectAtIndexPath(indexPath) {
            cell.configureWithShow(show)
        }

        return cell
    }
}
```

```
class ShowListCell: UITableViewCell {
    func configureWithShow(show: Show) {
        titleLabel.text = cell.name
        if let posterURL = NSURL(string: show.posterPath) {
            posterImageView.hnk_setImageFromURL(posterURL)
        }

        if let firstAired = someFormatter.dateFromString(show.firstAired) {
            firstAiredLabel.text = firstAired
        }
    }
}
```

```
class ShowViewModel {
    var identifier: String { get }
    var name: String { get }
    var posterURL: NSURL? { get }
    var firstAiredFormatted: String { get }

    init(show: Show)
}
```

```
class ShowListCell: UITableViewCell {
    func configureWithViewModel(viewModel: ShowViewModel) {
        titleLabel.text = viewModel.name

        if let posterURL = viewModel.posterURL {
            posterImageView.hnk_setImageFromURL(posterURL)
        }

        firstAiredLabel.text = viewModel.firstAiredFormatted
    }
}
```

```
class ShowListViewController: UIViewController, UICollectionViewDataSource {
    func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int {
        return Int(fetchedResultsController.sections.count)
    }

    func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        if let info = fetchedResultsController.sections[section] {
            return info.numberOfObjects
        }

        return 0
    }

    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier, forIndexPath: indexPath) as ShowCell

        if let show = fetchedResultsController.objectAtIndexPath(indexPath) {
            let viewModel = ShowViewModel(show)
            cell.configureWithViewModel(viewModel)
        }

        return cell
    }
}
```

```
class ShowListViewModel {
    var numberOfSections: Int { get }

    func numberOfShowsInSection(section: Int) -> Int
    func viewModelAtIndexPath(indexPath: NSIndexPath) -> ShowViewModel
}
```

```
class ShowListViewController: UIViewController, UICollectionViewDataSource {
    func numberOfSectionsInCollectionView(collectionView: UICollectionView) -> Int {
        return viewModel.numberOfSections
    }

    func collectionView(collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return viewModel.numberOfRowsInSection(section)
    }

    func collectionView(collectionView: UICollectionView, cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier, forIndexPath: indexPath) as ShowCell

        if let show = viewModel.viewModelAtIndexPath(indexPath)
            cell.configureWithViewModel(show)
        }

        return cell
    }
}
```

```
class ShowViewModel {
    var identifier: String { get }
    var name: Observable<String> { get }
    var posterURL: Observable<NSURL?> { get }
    var firstAiredFormatted: Observable<String> { get }

    init(show: Show)
}
```

```
class ShowListCell: UITableViewCell {
    func configureWithViewModel(viewModel: ShowViewModel) {
        viewModel.overview.afterChange += { [unowned self] in titleLabel.text = $1 }
        viewModel.posterImageURL.afterChange += { [unowned self] in self.posterImageView.hnk_setImageFromURL($1) }
        viewModel.firstAiredFormatted.afterChange += { [unowned self] in self.firstAiredLabel.text = $1 }
    }
}
```

@ksmandersen