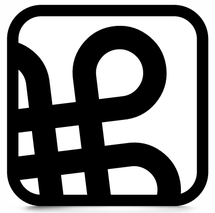


Macoun

Der Proto-Koller

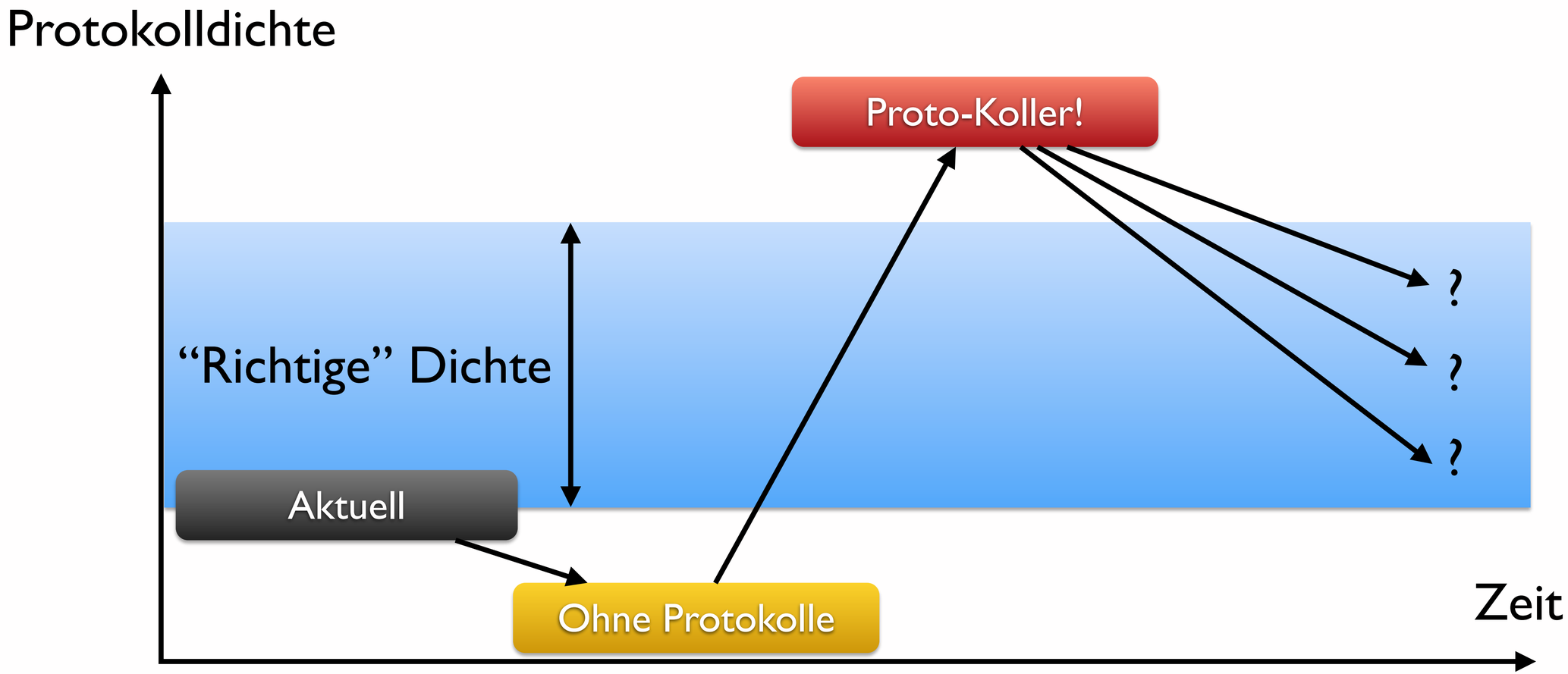
Tammo Freese
XING AG

Protokolle

- In Objective-C: “Wenige” Protokolle
  - Foundation (iOS 10): 356 Klassen, 26 Protokolle
- Swift: Deutlich mehr Protokolle
  - Typ Int (Swift 3): >=13 Protokolle

Setzen wir Protokolle bisher zu sparsam ein?

Modell


Protokolldichte
Proto-Koller!
?
?
?
“Richtige” Dichte
Aktuell
Ohne Protokolle
Zeit

Aufbau

- Beispiel
  - ohne Protokolle
  - im Proto-Koller
- Nebenwirkungen des Proto-Kollers
- Der Proto-Koller und Swift

Beispiel ohne Protokolle

Collection View Tools

- Modell für Inhalt einer Collection View
- Data Source für das Modell
- Updates zwischen zwei Modellen berechnen
- Updates auf Collection View ausführen

(vereinfachtes Beispiel)

Collection Model

- Modelliert den Inhalt einer Collection View
- Erzeugung durch änderbare Unterklasse

Collection Model

```
@interface TWOCollectionModel : NSObject <NSCopying, NSMutableCopying>

- (NSInteger)numberOfItemsInSection:(NSInteger)section;
- (NSString *)cellIdentifierAtIndexPath:(NSIndexPath *)indexPath;
- (id)cellModelAtIndexPath:(NSIndexPath *)indexPath;

@end

@interface TWOMutableCollectionModel : TWOCollectionModel

- (void)addCellWithIdentifier:(NSString *)reuseIdentifier model:(id)model;

@end
```

Collection Data Source

- Stellt ein Collection Model als Data Source bereit
- Zellkonfiguration durch Überschreiben einer Methode in Unterklassen

Collection Data Source

```
@interface TWOCollectionDataSource : NSObject <UICollectionViewDataSource>

@property (nonatomic, nullable, copy) TWOCollectionModel *collectionModel;

- (void)configureCell:(UICollectionViewCell *)cell
          atIndexPath:(NSIndexPath *)indexPath
            withModel:(id)model;

@end
```

Verwendung (1)

```
@interface DataSource : TWOCollectionDataSource
@end

@implementation DataSource

- (void)configureCell:(UICollectionViewCell *)cell
          atIndexPath:(NSIndexPath *)indexPath
            withModel:(id)model {
    ((Cell *)cell).text = model;
}

@end
```

Verwendung (2)

```
- (void)viewDidLoad {
    [super viewDidLoad];

    TWOMutableCollectionModel *model = [TWOMutableCollectionModel new];
    [model addCellWithIdentifier:@"Cell" model:@"A"];

    self.dataSource = [[DataSource alloc] init];
    self.dataSource.collectionModel = model;
    self.collectionView.dataSource = self.dataSource;
}
```

Collection Updates

- Modelliert die Änderungen von einem Modell zu einem anderen
- Erzeugung durch Klassenmethode
- Anwendung auf Collection View über Methode

Collection Updates

```
@interface TWOCollectionUpdates : NSObject

@property (nonatomic, readonly) NSArray *indexPathsForInsertedItems;
@property (nonatomic, readonly) NSArray *indexPathsForDeletedItems;
@property (nonatomic, readonly) NSArray *indexPathsForReloadedItems;
@property (nonatomic, readonly) NSDictionary *indexPathsForMovedItems;

+ (instancetype)updatesFromModel:(TWOCollectionModel *)fromModel
                         toModel:(TWOCollectionModel *)toModel;

- (void)performOnCollectionView:(UICollectionView *)collectionView;

@end
```

Verwendung (3)

```
[self.collectionView performBatchUpdates:^{
    TWOCollectionModel *fromModel = self.dataSource.collectionModel;
    TWOCollectionUpdates *updates = [TWOCollectionUpdates
                                     updatesFromModel:fromModel
                                     toModel:toModel];
    self.dataSource.collectionModel = toModel;
    [updates performOnCollectionView:self.collectionView];
} completion:nil];
```

Beispiel im Proto-Koller

Der Proto-Koller

- Protokolle statt Klassen für eigene Typen
- Problem Objekterzeugung: Erstmal über Klassenmethoden

Collection Model

- Modelliert den Inhalt einer Collection View
- Erzeugung durch ~~änderbare Unterklasse~~ *Klassenmethode mit neuem Protokoll*

Collection Model

```
@protocol TWOCollectionModel <NSObject>
- (NSInteger)numberOfItemsInSection:(NSInteger)section;
- (NSString *)cellIdentifierAtIndexPath:(NSIndexPath *)indexPath;
- (id)cellModelAtIndexPath:(NSIndexPath *)indexPath;
@end

@protocol TWOCollectionModelBuilder <NSObject>
- (void)addCellWithIdentifier:(NSString *)reuseIdentifier model:(id)model;
@end

@interface TWOCollectionTools : NSObject
+ (id<TWOCollectionModel>)modelWithBuilder:
        (void (^)(id<TWOCollectionModelBuilder>))builder;
@end
```

Collection Data Source

- Stellt ein Collection Model als Data Source bereit
- Zellkonfiguration durch ~~Überschreiben einer Methode in Unterklassen~~ *neues Protokoll als Property*
- *Erzeugung über Klassenmethode*

Collection Data Source

```
@protocol TWOCollectionCellConfigurator <NSObject>
- (void)configureCell:(UICollectionViewCell *)cell
          atIndexPath:(NSIndexPath *)indexPath
            withModel:(id)model;
@end

@protocol TWOCollectionDataSource <UICollectionViewDataSource>

@property (nonatomic, weak) id<TWOCollectionModel> collectionModel;
@property (nonatomic, weak) id<TWOCollectionCellConfigurator> cellConfigu;

@end
```

Verwendung (1)

```
@interface ViewController () <TWOCollectionCellConfigurator>
@end

…

// Im ViewController:
- (void)configureCell:(UICollectionViewCell *)cell
          atIndexPath:(NSIndexPath *)indexPath
            withModel:(id)model {
    ((Cell *)cell).text = model;
}
```

Verwendung (2)

```
- (void)viewDidLoad {
    [super viewDidLoad];

    self.model = [TWOCollectionTools modelWithBuilder:
                  ^(id<TWOCollectionModelBuilder> builder) {
        [builder addCellWithIdentifier:@"Cell" model:@"A"];
    }];
    self.dataSource = [TWOCollectionTools dataSource];
    self.dataSource.collectionModel = self.model;
    self.dataSource.cellConfigurator = self;
    self.collectionView.dataSource = self.dataSource;
}
```

Collection Updates

- Modelliert die Änderungen von einem Modell zu einem anderen
- Erzeugung durch Klassenmethode
- Anwendung auf ~~Collection View~~ *Target (neues Protokoll)* über *Klassen*methode

Collection View Updates

```
@protocol TWOCollectionUpdates <NSObject>

@property (nonatomic, readonly) NSArray *indexPathsForInsertedItems;
@property (nonatomic, readonly) NSArray *indexPathsForDeletedItems;
@property (nonatomic, readonly) NSArray *indexPathsForReloadedItems;
@property (nonatomic, readonly) NSDictionary *indexPathsForMovedItems;

@end

@interface TWOCollectionTools : NSObject
+ (id<TWOCollectionUpdates>)updatesFromModel:(id<TWOCollectionModel>)fromModel
                                     toModel:(id<TWOCollectionModel>)toModel;
+ (void)performUpdates:(id<TWOCollectionUpdates>)updates
              onTarget:(id<TWOCollectionUpdatesTarget>)target;
@end
```

Collection Updates Target

```
@protocol TWOCollectionUpdatesTarget <NSObject>

- (void)insertItemsAtIndexPaths:(NSArray<NSIndexPath *> *)indexPaths;
- (void)deleteItemsAtIndexPaths:(NSArray<NSIndexPath *> *)indexPaths;
- (void)reloadItemsAtIndexPaths:(NSArray<NSIndexPath *> *)indexPaths;
- (void)moveItemAtIndexPath:(NSIndexPath *)indexPath
                toIndexPath:(NSIndexPath *)newIndexPath;

@end

@interface UICollectionView (TWOCollectionTools) <TWOCollectionUpdatesTarget>
@end
```

Verwendung (3)

```
[self.collectionView performBatchUpdates:^{
    id<TWOCollectionModel> fromModel = self.model;
    id<TWOCollectionUpdates> updates = [TWOCollectionTools
                                        updatesFromModel:fromModel
                                        toModel:toModel];
    self.model = toModel;
    self.dataSource.collectionModel = toModel;
    [TWOCollectionTools performUpdates:updates
                              onTarget:self.collectionView];
} completion:nil];
```

Proto-Koller extrem: Klassenmethoden vermeiden

- TWOCollectionTools: Sammelbecken für Erzeuger-Methoden
- Auch möglich: Simulieren von “Klassenmethoden für Protokolle”

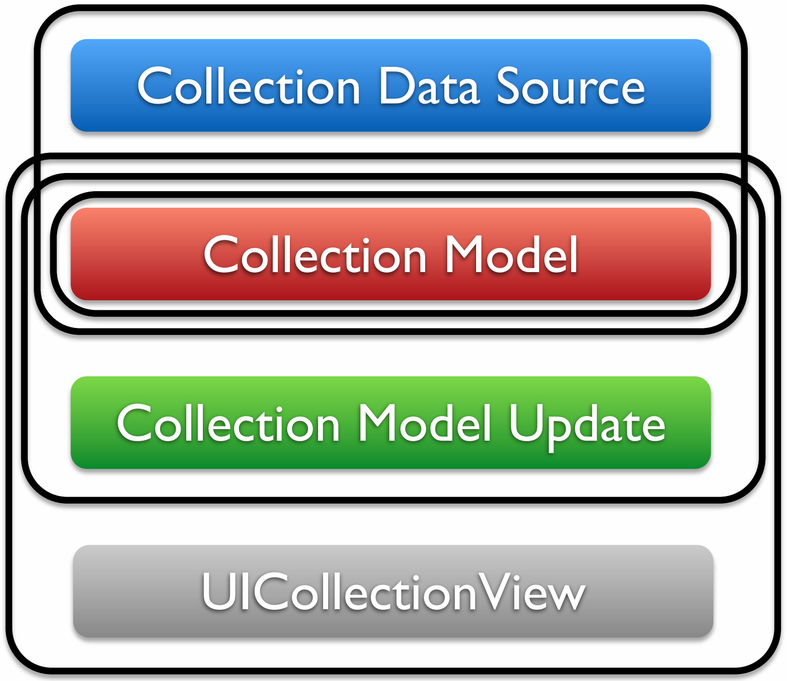
Klassenmethoden vermeiden

```
@protocol TWOCollectionDataSourceCreator <NSObject>
- (id<TWOCollectionDataSource>)dataSource;
@end
extern id<TWOCollectionDataSourceCreator> TWOCollectionDataSource;

...

// Verwendung im Controller
self.dataSource = [TWOCollectionDataSource dataSource];

// Alternativ für Testbarkeit: dataSourceCreator-Property einführen
self.dataSource = [self.dataSourceCreator dataSource];
```

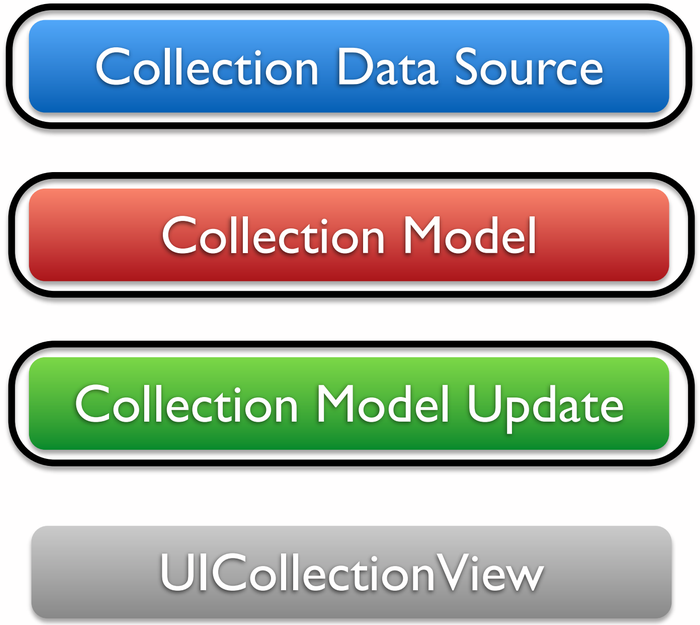
Nebenwirkungen des Proto-Kollers

Entkopplung

Ohne Protokolle

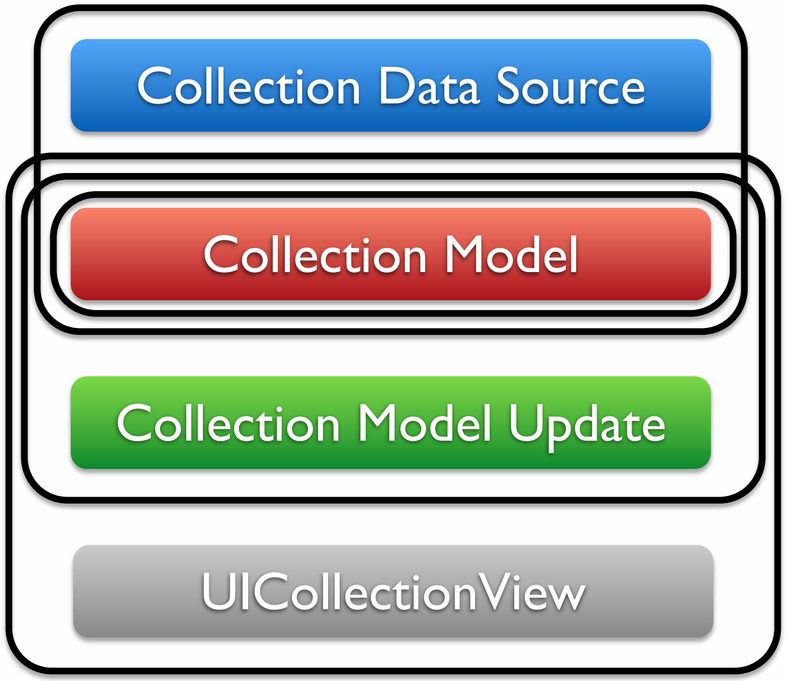


Proto-Koller

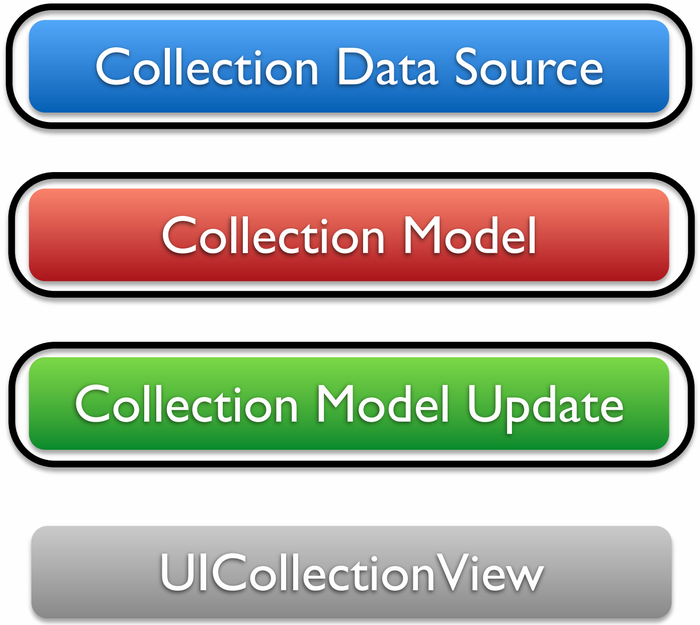


Testen

Ohne Protokolle

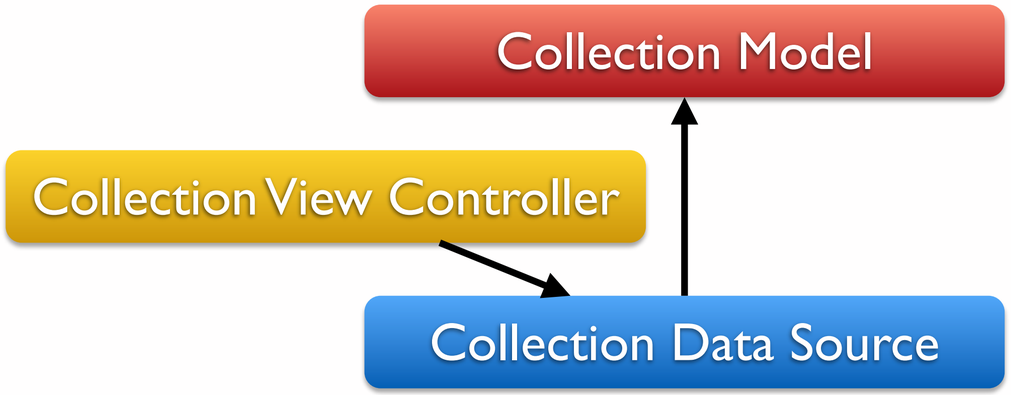


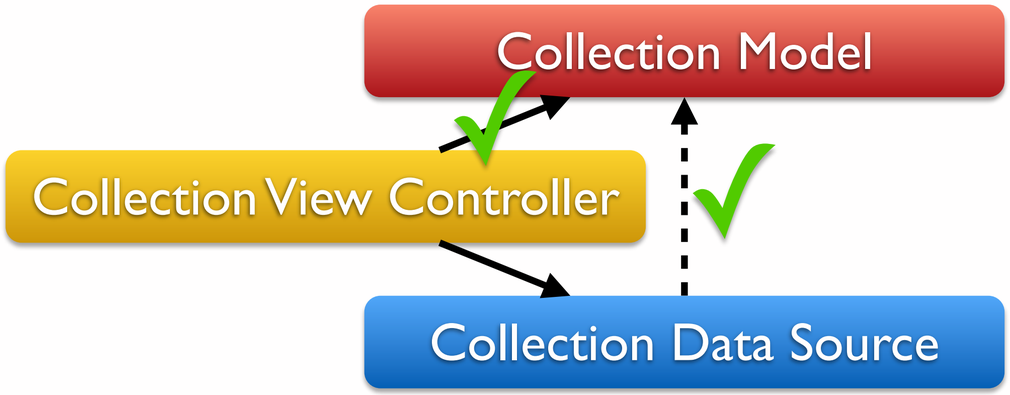
Proto-Koller



Memory Management

- Gefahr bei Protokollen: strong reference cycle
- Daher bei Properties: weak bevorzugen
- Nebenwirkung: Zusätzliche strong Properties beim Verwender





Weitere Nebenwirkungen

- Information Hiding: Implementierende Klasse kann versteckt werden
- Protokoll verhindert (na gut, erschwert) Subclassing
- Leider keine Generics auf Protokollen
- Auto-Completion in Xcode 8 geht nicht auf Protokoll-Properties

Der Proto-Koller und Swift

Der Proto-Koller und Swift

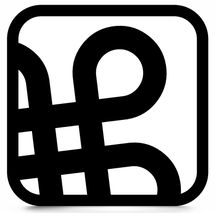
- Protokolle auch für struct und enum!
- Protocol Extensions!
- Swift-Protokolle: Keine Generics, aber associated types!
- Auto-Completion in Xcode 8 geht auf Protokoll-Properties!

Setzen wir Protokolle bisher zu sparsam ein?

Ich denke ja.

Fragen?

Vielen Dank!



Macoun