andrena OBJECTS

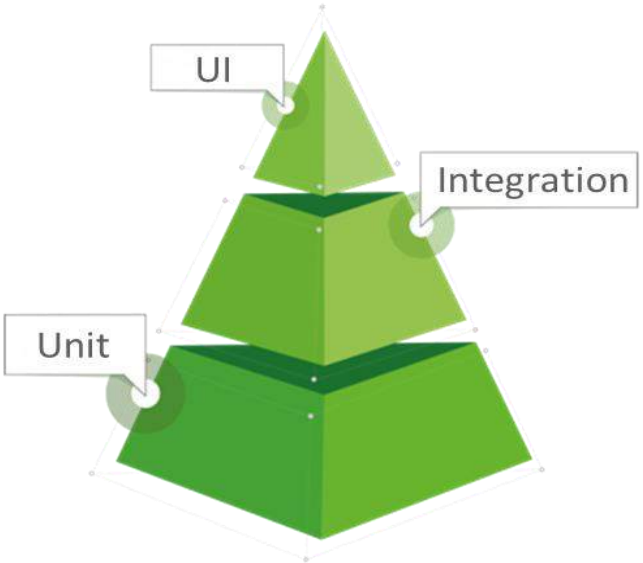
Experts in agile software engineering

JavaScript: Von der Eistüte zur Testpyramide

Eva Ziebarth, Heinrich Franz, Sylvia Raab

Projekt Setup

- Entwicklung einer UI Anwendnung
  - AngularJS
  - JavaScript/TypeScript
  - Layout Framework



Agenda

- Basics
- Bad Practices
- Generelles Vorgehen
- Refactoring
- Fazit

Basics

Herausforderung

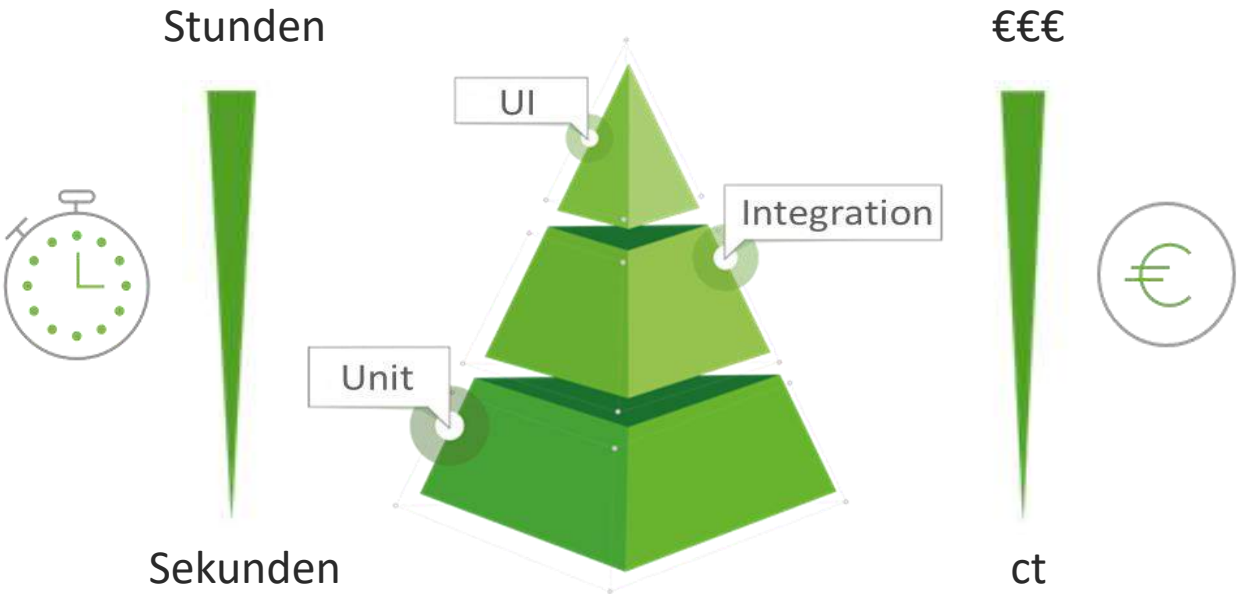
- Bestehendes System produktiv halten
- Bestehende Funktionalität erhalten oder verbessern
- Anforderung PO:

  Feature getriebene Software Entwicklung

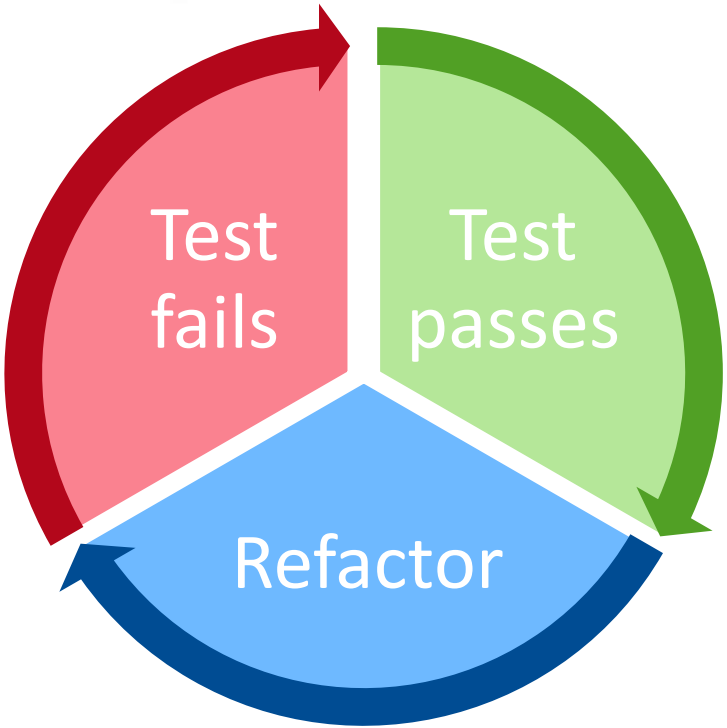
  vs.

  Innere Qualität steigern

Testpyramide



Test Driven Development (TDD)

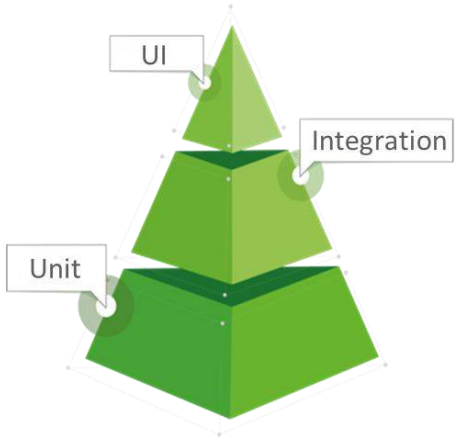


Clean Code

- Aussagekräftige Namen
- Wenige Argumente in einer Methode
- Kaum Kommentare

OO-Prinzipien*

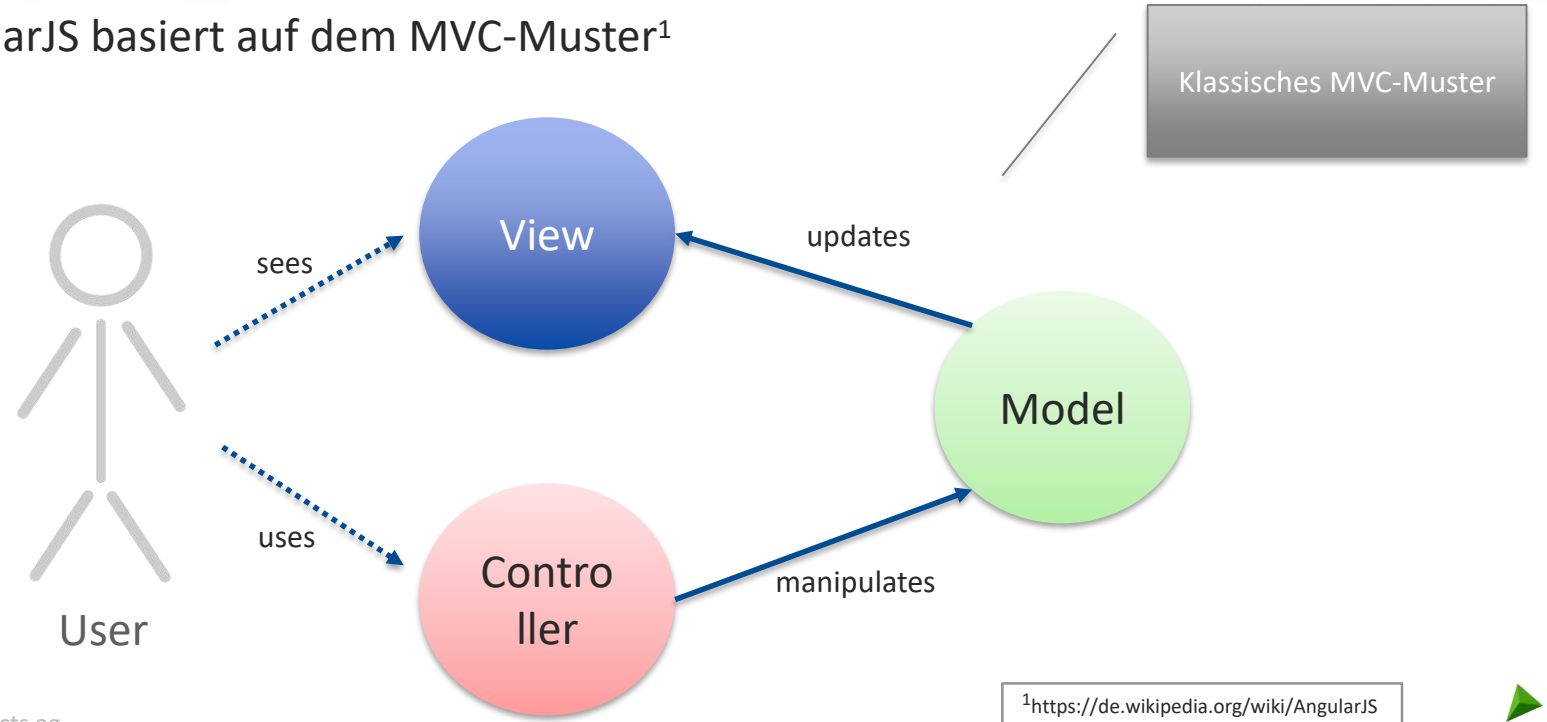
- Single Responsibility
  → Für die Änderung einer Klasse gibt es immer nur einen Grund
- Open Closed Principle
  → Software-Einheiten sind erweiterbar, ohne dabei ihr Verhalten zu ändern
- Acyclic Dependencies Principle
  → Zyklen-freie Abhängigkeiten zwischen Modulen
- Typisierung
- Separation of Concerns



___
* auszugsweise

AngularJS

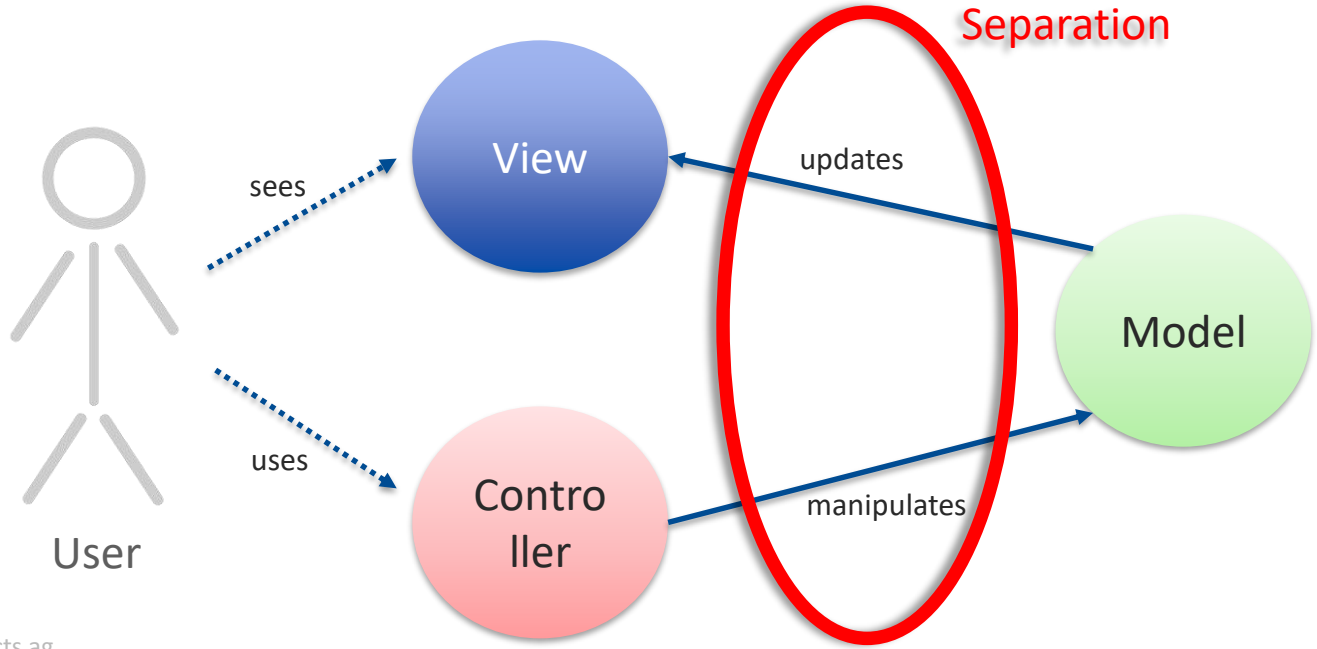
- AngularJS basiert auf dem MVC-Muster[1]



[1] https://de.wikipedia.org/wiki/AngularJS

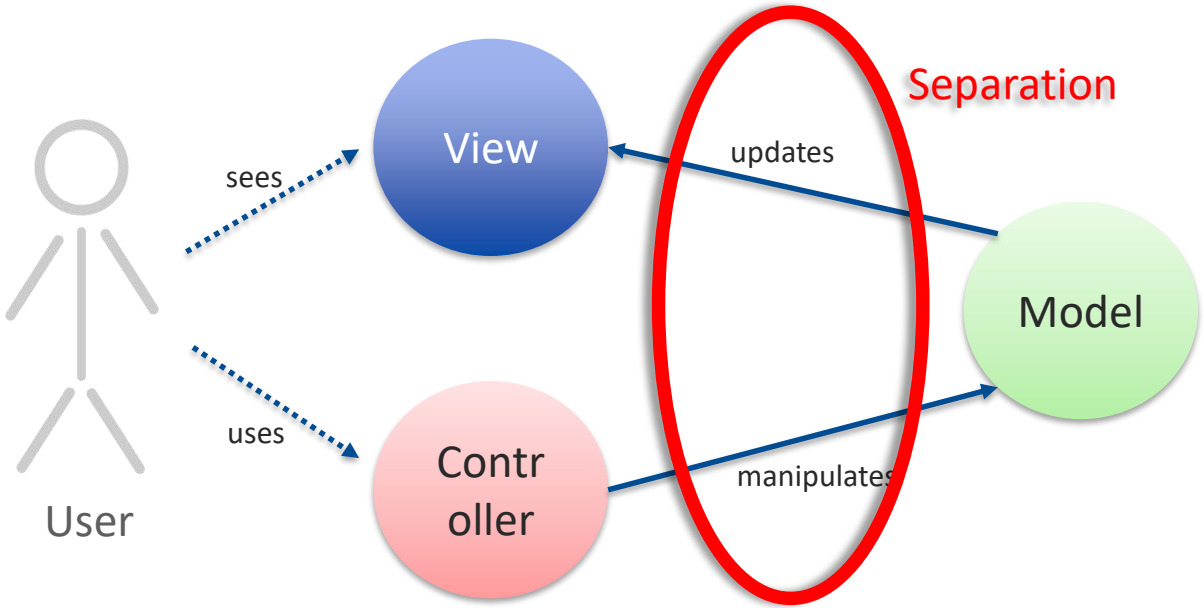
AngularJS mit klassischem MVC-Muster

- Separation of concerns:



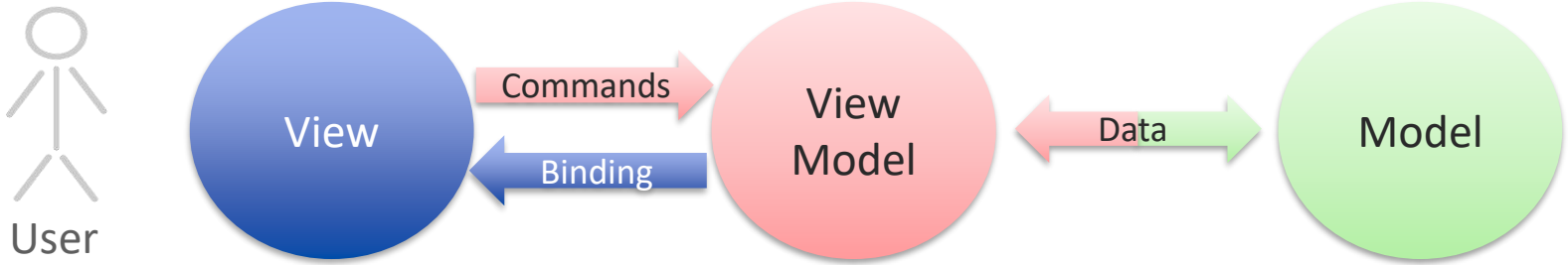
AngularJS

- AngularJS-Framework bietet allerdings 2-way Data-Binding



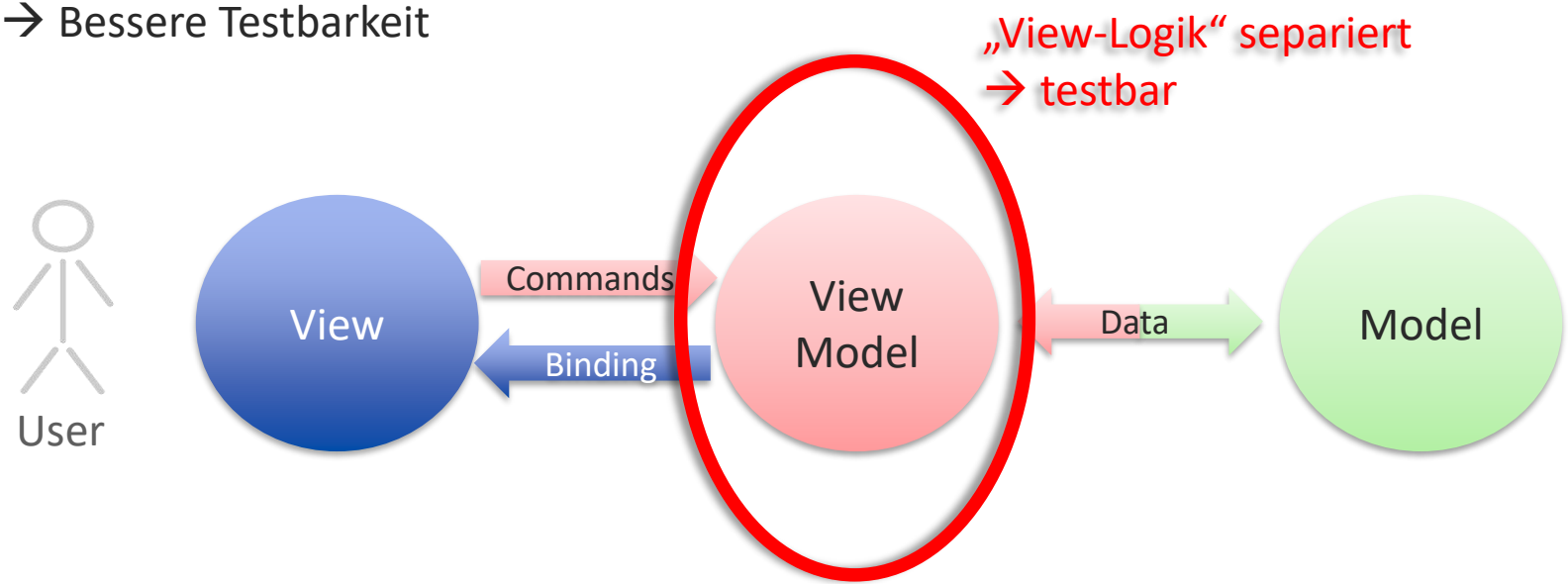
AngularJS mit MVVM-Muster

- Data-Binding nach MVVM-Muster
- Keine enge Kopplung zwischen View und Model



AngularJS mit MVVM-Muster

- Separation of concerns
- → Bessere Testbarkeit



Bad Practices

Beispiel

inject $scope

Beispiel – inject $scope

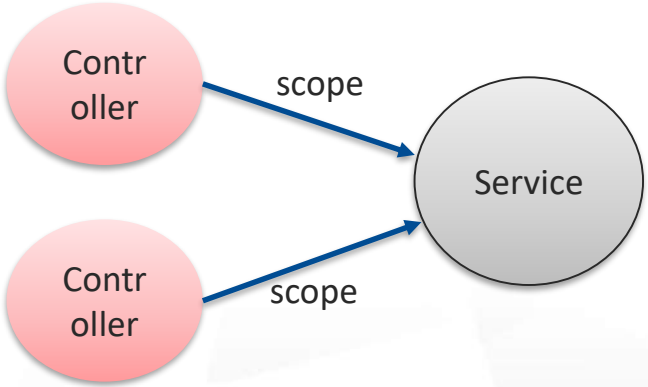
- Problem:
  scope-Objekt (Model)
  untypisiert
- „Kontrolle“ über scope liegt bei Framework
- Testbarkeit nur über Controller durch „reinreichen“

```
<div ng-app="myApp" ng-controller="myCtrl">
    <input ng-model="name">
    <h1>My name is {{name}}</h1>
</div>


var app = angular.module('myApp', []);
app.controller('myCtrl', function($scope) {
    $scope.name = "John Doe";
});
```

Beispiel – inject $scope

Mehrere Views synchron halten:

- scope von Controller an angular-Services weitergeben **(Achtung: Singletons!)**



→ Testbarkeit problematisch

Beispiel

Open-Closed-Prinzip verletzt

```
public validatePage() : void {
    if (!this.timePeriod.isValid ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FromTimeErrorLabel";
    }
    else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "TimeSpanInPastNotAllowed";
    }
    else if (!this.description.predefinedDescription) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "SelectValidDescription";
    }
    else if (!this.description.isSelectionFinished() ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "DetailMissingLabel";
    }
    else if (!this.validateTitle()) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FaultyDescriptionMessage";
    }
    else {
        this.dialogConfig.formHint.state = "withoutErrors";
        this.dialogConfig.formHint.message = "";
    }
    this.isValid = this.dialogConfig.formHint.state === "withoutErrors";
}
```

Switch case + code-Duplikationen

Beispiel

Vermischung von Code und Style

```
<html>
. . .
    <dogbox id="boxOfLittleDogs"
            is-visible="dogController.list.length !===0 &&
                        (dogController.list.filter(dog => dog.type === DogType.beagle).length !==0
                            || dogController.list.filter(dog => dog.type === DogType.yorkshireTerrier).length !== 0
                            || dogController.list.filter(dog => dog.type === DogType.chihuaua).length !== 0
                            || dogController.list.filter(dog => dog.type === DogType.poodle).length !== 0
                            || dogController.list.filter(dog => dog.age <= 12).length !== 0)"
            dogs="dogController.list | littleDogs">
    </dogbox>
. . .
</html>
```

Javascript Code in HTML => nur testbar via UI-Tests
oder sehr komplizierten Unit-Tests (hier: viele Cases)

```
protected createDogListRenderer() {
            return new GridColumn("Name", "Description", (data, type, full: Dog) => {
                let creationDate: string = this.$filter('date')(full.creationTime, 'medium');
                let dogItemStyle = "";
                if (full.character !== null) {
                    dogItemStyle += "border-color: " + this.getHighlightColour(full.character) + ";";
                }

                return
                    `<div class="dogItem" style="` + dogItemStyle + `">` +
                      `<div class="firstRow">` +
                       `<svg-icon data-icon="` + full.icon + `" data-size="24"></svg-icon>` +
                      `</div>` +
                      `<div class="dogIdentifier">` +
                         `<span>` + full.type + `</span>` +
                         `<span>` + full.name + `</span>` +
                       `</div>` +
                       `<div class="timestamp">` +
                            full.birthDate +
                       `</div>` +
                    `</div>`
            }
}
```

HTML in Javascript Code, als einfacher String (nicht als HTML element o.ä.) => aussagekräftige Tests schwer

Beispiel

OO-Prinzipien verletzt

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```

Variablen, die von einer verwendeten Bibliothek gebraucht, aber hier nicht benutzt werden

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```

Was ist mit data[1] und data[2]???

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```

Warum muss das hier gemacht werden?

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```

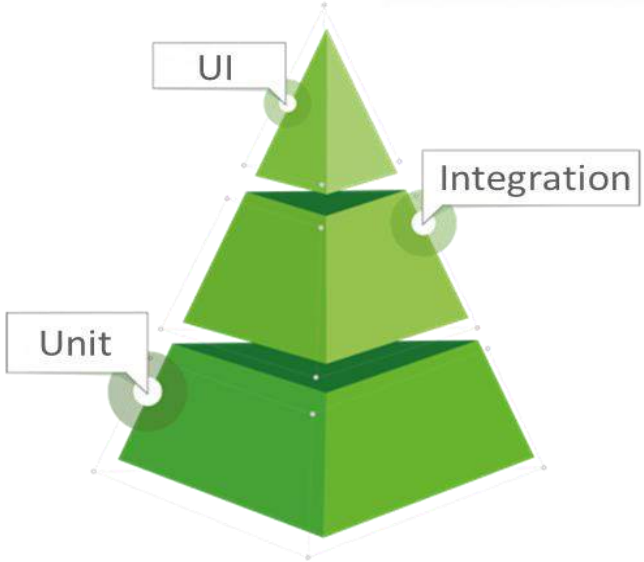

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```


```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```


Generelles Vorgehen

Generelles Vorgehen

- Parallele Neuentwicklung
  - TDD
  - Vergleich zur bestehenden App
- Umsetzung MVVM
- Alles typisiert
- Framework Aufrufe kapseln



Refactoring

Beispiel

inject $scope

Beispiel – inject $scope

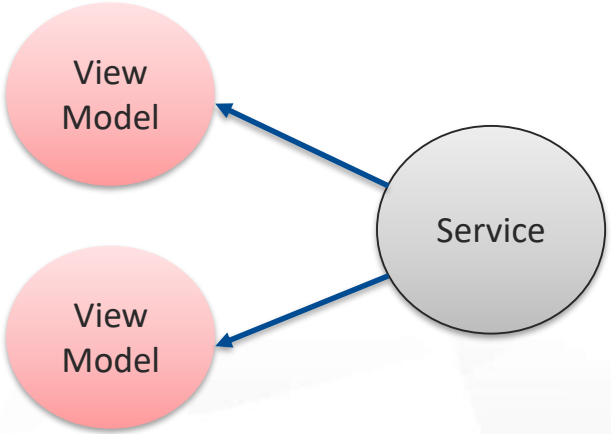
- ViewModel
  ist ngController und enthält „Model“ der View

=> Kein $scope in ngController mehr notwendig

```
<div ng-app="myApp" ng-controller="myVM">
    <input ng-model="myVM.name">
    <h1>My name is {{myVM.name}}</h1>
</div>


class MyTestViewModel {
    public name;
}

let app = angular.module('myApp', []);
app.controller('myVM', MyTestViewModel);
```

Beispiel – inject $scope

- Angular-Service (Singleton), in ViewModels der Ansichten mitgeben um mehrere Views synchron zu halten



=> Kein $scope in ngController und Service mehr notwendig

Beispiel

Open-Closed-Prinzip verletzt

```
public doFormValidation() : void {
    if (!this.timePeriod.isValid ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FromTimeErrorLabel";
    }
    else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "TimeSpanInPastNotAllowed";
    }
    else if (!this.description.predefinedDescription) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "SelectValidDescription";
    }
    else if (!this.description.isSelectionFinished() ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "DetailMissingLabel";
    }
    else if (!this.validateTitle()) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FaultyDescriptionMessage";
    }
    else {
        this.dialogConfig.formHint.state = "withoutErrors";
        this.dialogConfig.formHint.message = "";
    }
    this.isValid = this.dialogConfig.formHint.state === "withoutErrors";
}
```


```
export class FormValidityState {
    public static invalid: string = "withErrors";
    public static valid: string = "withoutErrors";
}

export interface IFormValidationResult {
    state: ValidationState;
    message: string;
}

export class FormValidationResult implements IFormValidationResult{
    constructor(state: ValidationState, message: string) {
        this.mState = state;
        this.mMessage = message;
    }

    public get state(): ValidationState {
        . . .
    }

    public get message(): string {
        . . .
    }
    . . .
}
```

Einführung neuer Interfaces und Klassen
⇒ nicht-typisierte API des eingebundenen Fremdsystems typisieren

```
export class FormValidityState {
    public static invalid: string = "withErrors";
    public static valid: string = "withoutErrors";
}
```

Definition so gewählt, dass FormValidityState benutzt werden kann wie ein Enum

```
export interface IFormValidationResult {
    state: ValidationState;
    message: string;
}

export class FormValidationResult implements IFormValidationResult{
    constructor(state: ValidationState, message: string) {
        this.mState = state;
        this.mMessage = message;
    }

    public get state(): ValidationState {
        . . .
    }

    public get message(): string {
        . . .
    }
    . . .
}
```


```
export class FormValidityState {
    public static invalid: string = "withErrors";
    public static valid: string = "withoutErrors";
}

export interface IFormValidationResult {
    state: ValidationState;
    message: string;
}

export class FormValidationResult implements IFormValidationResult{
    constructor(state: ValidationState, message: string) {
        this.mState = state;
        this.mMessage = message;
    }

    public get state(): ValidationState {
        . . .
    }

    public get message(): string {
        . . .
    }
    . . .
}
```

Einführung eines Datentyps, um Zuweisung auf Objektebene anstatt Feldebene machen zu können.

```
export class FormValidityState {
    public static invalid: string = "withErrors";
    public static valid: string = "withoutErrors";
}
```

```
export interface IFormValidationResult {
    state: ValidationState;
    message: string;
}
```

Einführung eines Datentyps, um Zuweisung auf Objektebene anstatt Feldebene machen zu können.

```
export class FormValidationResult implements IFormValidationResult{
    constructor(state: ValidationState, message: string) {
        this.mState = state;
        this.mMessage = message;
    }

    public get state(): ValidationState {
        . . .
    }

    public get message(): string {
        . . .
    }
    . . .
}
```

```
this.dialogConfig.formHint.state = "withErrors";
this.dialogConfig.formHint.message = "FromTimeErrorLabel";
```

```
this.dialogConfig.formHint
   = new FormValidationResult(FormValidityState.invalid,"FromTimeErrorLabel")
```

```
public doFormValidation() : void {
    if (!this.timePeriod.isValid ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FromTimeErrorLabel";
    }
    else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "TimeSpanInPastNotAllowed";
    }
    else if (!this.description.predefinedDescription) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "SelectValidDescription";
    }
    else if (!this.description.isSelectionFinished() ) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "DetailMissingLabel";
    }
    else if (!this.validateTitle()) {
        this.dialogConfig.formHint.state = "withErrors";
        this.dialogConfig.formHint.message = "FaultyDescriptionMessage";
    }
    else {
        this.dialogConfig.formHint.state = "withoutErrors";
        this.dialogConfig.formHint.message = "";
    }
    this.isValid = this.dialogConfig.formHint.state === "withoutErrors";
}
```


```
public doFormValidation = () : void => {
      let formValidationResult: IFormValidationResult;
      if (!this.TimeSpan.isValid) {
          formValidationResult = new FormValidationResult(FormValidityState.invalid, "FromTimeErrorLabel");
      }
      else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
          formValidationResult = new FormValidationResult(FormValidityState.invalid, "TimeSpanInPastNotAllowed");
      }
      else if (!this.description.predefinedDescription) {
          formValidationResult = new FormValidationResult(FormValidityState.invalid, "SelectValidDescription");
      }
      else if (!this.description.isSelectionFinished()) {
          formValidationResult = new FormValidationResult(FormValidityState.invalid, "DetailMissingLabel");
      }
      else if (!this.validateTitle()) {
          formValidationResult = new FormValidationResult(FormValidityState.invalid, "FaultyDescriptionMessage");
      }
      else {
          formValidationResult = new FormValidationResult(FormValidityState.valid, "");
      }
      this.dialogConfig.formHint = formValidationResult;
      this.isValid = this.dialogConfig.formHint.state === FormValidityState.valid;
  }
```


```
public doFormValidation = (): void => {
        this.dialogConfig.formHint = this.validatePage();
        this.isValid = this.dialogConfig.formHint.state === FormValidityState.valid;
    }

    private validatePage = (): IFormValidationResult => {
        let formValidationResult: IFormValidationResult;
        if (!this.TimeSpan.isValid) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "FromTimeErrorLabel");
        }
        else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "TimeSpanInPastNotAllowed");
        }
        else if (!this.description.predefinedDescription) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "SelectValidDescription");
        }
        else if (!this.description.isSelectionFinished()) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "DetailMissingLabel");
        }
        else if (!this.validateTitle()) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "FaultyDescriptionMessage");
        }
        else {
            formValidationResult = new FormValidationResult(FormValidityState.valid, "");
        }

        return formValidationResult;
    }
```

Extrahiere Methode validatePage, die formHint berechnet

```
public doFormValidation = (): void => {
        this.dialogConfig.formHint = this.validatePage();
        this.isValid = this.dialogConfig.formHint.state === FormValidityState.valid;
    }

    private validatePage = (): IFormValidationResult => {
        let formValidationResult: IFormValidationResult;
        if (!this.TimeSpan.isValid) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "FromTimeErrorLabel");
        }
        else if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "TimeSpanInPastNotAllowed");
        }
        else if (!this.description.predefinedDescription) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "SelectValidDescription");
        }
        else if (!this.description.isSelectionFinished()) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "DetailMissingLabel");
        }
        else if (!this.validateTitle()) {
            formValidationResult = new FormValidationResult(FormValidityState.invalid, "FaultyDescriptionMessage");
        }
        else {
            formValidationResult = new FormValidationResult(FormValidityState.valid, "");
        }

        return formValidationResult;
    }
```

Eliminieren einer lokalen Variablen mit sehr langer Lebenszeit => Vereinfachung des Ablaufs

```
public doFormValidation = (): void => {
    this.dialogConfig.formHint = this.validatePage();
    this.isValid = this.dialogConfig.formHint.state === FormValidityState.valid;
}

private validatePage = (): IFormValidationResult => {
    if (!this.TimeSpan.isValid) {
        return new FormValidationResult(FormValidityState.invalid, "FromTimeErrorLabel");
    }

    if (DateHelper.isDateInPast(this.fromTime) || DateHelper.isDateInPast(this.toTime)) {
        return new FormValidationResult(FormValidityState.invalid, "TimeSpanInPastNotAllowed");
    }

    if (!this.description.predefinedDescription) {
        return new FormValidationResult(FormValidityState.invalid, "SelectValidDescription");
    }

    if (!this.description.isSelectionFinished()) {
        return new FormValidationResult(FormValidityState.invalid, "DetailMissingLabel");
    }

    if (!this.validateTitle()) {
        return new FormValidationResult(FormValidityState.invalid, "FaultyDescriptionMessage");
    }

    return new FormValidationResult(FormValidityState.valid, "");
}
```


```
private isTimePeriodValid = (): boolean => { . . . }
private isDescriptionPredefined = (): boolean => {. . .}

private validatePage = (): IFormValidationResult => {
    if (!this.TimeSpan.isValid) {
        return new FormValidationResult(FormValidityState.invalid, "FromTimeErrorLabel");
    }

    if (!this.isTimePeriodValid()) {
        return new FormValidationResult(FormValidityState.invalid, "TimeSpanInPastNotAllowed");
    }

    if (!this.isDescriptionPredefined()) {
        return new FormValidationResult(FormValidityState.invalid, "SelectValidDescription");
    }

    if (!this.isDescriptionSelected()) {
        return new FormValidationResult(FormValidityState.invalid, "DetailMissingLabel");
    }

    if (!this.validateTitle()) {
        return new FormValidationResult(FormValidityState.invalid, "FaultyDescriptionMessage");
    }

    return new FormValidationResult(FormValidityState.valid, "");
}
```

Einführen von Methoden für Validierungen (wo noch nicht geschehen)

Einführen einer Klasse, die Validierungsregeln repräsentiert

```
export interface IFormValidationRule{
    getResult: () => IFormValidationResult;
    fieldName: string;
    message: string;
    isValid: () => boolean;
}

export class FormValidationRule implements IFormValidationRule {
        constructor(fieldName: string, message: string, isValid: () => boolean) {
            . . .
        }

         public getResult = (): IFormValidationResult => {
            return new UiFrameworkFormValidationError(this.fieldName, this.mMessage);
        }

        public isValid = (): boolean => {
            . . .
        }

        public get message(): string {
            . . .
        }

        public get fieldName(): string {
            . . .
        }
    }
```

```
export interface IFormValidationRule{
    getResult: () => IFormValidationResult;
    fieldName: string;
    message: string;
    isValid: () => boolean;
}
```

Methode, die Datentyp zurückgibt, der vom angeschlossenen Fremdframework erwartet wird

```
export class FormValidationRule implements IFormValidationRule {
        constructor(fieldName: string, message: string, isValid: () => boolean) {
            . . .
        }

         public getResult = (): IFormValidationResult => {
            return new UiFrameworkFormValidationError(this.fieldName, this.mMessage);
        }

        public isValid = (): boolean => {
            . . .
        }

        public get message(): string {
            . . .
        }

        public get fieldName(): string {
            . . .
        }
    }
```

```
export interface IFormValidationRule{
    getResult: () => IFormValidationResult;
    fieldName: string;
    message: string;
    isValid: () => boolean;
}

export class FormValidationRule implements IFormValidationRule {
        constructor(fieldName: string, message: string, isValid: () => boolean) {
            . . .
        }

         public getResult = (): IFormValidationResult => {
            return new UiFrameworkFormValidationError(this.fieldName, this.mMessage);
        }

        public isValid = (): boolean => {
            . . .
        }

        public get message(): string {
            . . .
        }

        public get fieldName(): string {
            . . .
        }
    }
```

Zur Erinnerung:

```
export interface IFormValidationResult {
     state: ValidationState;
     message: string;
 }
```

```
export class FormValidationList {
        private validations: IFormValidationRule[];

        constructor() {
            ...
        }

        public add = (validation: IFormValidationRule): void => {
            if (!validation) {
                throw new TypeError("Validation cannot be null or undefined.");
            }

            this.validations.push(validation);
        }

        public areValid = (): boolean => {
            ...
        }

        public findInvalid = (): IFormValidationRule[] => {
            return this.validations.filter(validation => !validation.isValid());
        }

        public clear = (): void => {
            this.validations = [];
        }
    }
```

```
export interface IFormValidationRule{
    getResult: () => IFormValidationResult;
    fieldName: string;
    message: string;
    isValid: () => boolean;
}
```

Einführung einer ValidationList-Klasse, die sich um das Hinzufügen und Kombinieren der Validierungsregeln kümmert (TDD)

```
describe(„formValidationList",
        () => {
            describe("add",
                () => {
                    let validationList: FormValidationList;
                    let testValidation: IFormValidationRule;

                    beforeEach(() => {
                        initilizeTestData();
                    });

                    it("should add given validation to list",
                        () => {
                            validationList.add(testValidation);

                            expect(validationList.length).toBe(1);
                            expect(validationList[0]).toBe(testValidation);
                        });

                    it("should throw exception, if trying to add null",
                        () => {
                            . . .
                        });

                    . . .

                }); . . . });
```


```
export class FormValidationList {
        private validations: IFormValidationRule[];

        constructor() {
            ...
        }

        public add = (validation: IFormValidationRule): void => {
            if (!validation) {
                throw new TypeError("Validation cannot be null or undefined.");
            }

            this.validations.push(validation);
        }

        public areValid = (): boolean => {
            ...
        }

        public findInvalid = (): IFormValidationRule[] => {
            return this.validations.filter(validation => !validation.isValid());
        }

        public clear = (): void => {
            this.validations = [];
        }
    }
```

```
export interface IFormValidationRule{
     getResult: () => IFormValidationResult;
     fieldName: string;
     message: string;
     isValid: () => boolean;
 }
```

Einführung einer ValidationList-Klasse, die sich um das Hinzufügen und Kombinieren der Validierungsregeln kümmert (TDD)

```
private formValidationList: FormValidationList;

private initializeFormValidations = () => {
    this.formValidationList = new FormValidationList();
    this.formValidationList.add(new FormValidationRule("TimeSpan", "FromTimeErrorLabel", this.isTimePeriodValid));
    this.formValidationList.add(new FormValidationRule("TimeSpan", "TimeSpanInPastNotAllowed", this.isAnyDateInPast));
    this.formValidationList.add(new FormValidationRule("description", "SelectValidDescription", this.isDescriptionPredefined));
    this.formValidationList.add(new FormValidationRule("description", "DetailMissingLabel", this.isDescriptionSelected));
    this.formValidationList.add(new FormValidationRule("title", "DetailMissingLabel", this.validateTitle));
}

private validatePage = (): IFormValidationResult => {
    let isPageValid = this.formValidationList.areValid();
    if (isPageValid) {
        return new FormValidationResult(FormValidityState.valid, "");
    }

    return this.formValidationList.findFirstInvalid().getResult();
}

public doFormValidation = (): void => {
    this.dialogConfig.formHint = this.validatePage();
    this.isValid = this.formValidationList.areValid();
}
```

Muss nur einmal in Initialisierungsphase aufgerufen werden

```
private formValidationList: FormValidationList;

private initializeFormValidations = () => {
     this.formValidationList = new FormValidationList();
     this.formValidationList.add(new FormValidationRule("TimeSpan", "FromTimeErrorLabel", this.isTimePeriodValid));
     this.formValidationList.add(new FormValidationRule("TimeSpan", "TimeSpanInPastNotAllowed", this.isAnyDateInPast));
     this.formValidationList.add(new FormValidationRule("description", "SelectValidDescription", this.isDescriptionPredefined));
     this.formValidationList.add(new FormValidationRule("description", "DetailMissingLabel", this.isDescriptionSelected));
     this.formValidationList.add(new FormValidationRule("title", "DetailMissingLabel", this.validateTitle));
}

private validatePage = (): IFormValidationResult => {
     let isPageValid = this.formValidationList.areValid();
     if (isPageValid) {
         return new FormValidationResult(FormValidityState.valid, "");
     }

     return this.formValidationList.findFirstInvalid().getResult();
}

public doFormValidation = (): void => {
     this.dialogConfig.formHint = this.validatePage();
     this.isValid = this.formValidationList.areValid();
}
```

Neue Struktur der ursprünglichen Methode mit Validierungsliste

Nächste Schritte:

- ValidatePage-Funktion in formValidationList ziehen
- Funktionalitäten in eigenen angular-Service ziehen (validationService) und Service in ViewModel injecten
- . . .

Beispiel

Vermischung von Code und Style

```
<html>
. . .
    <dogbox id="boxOfLittleDogs"
            is-visible="dogController.list.length !===0 &&
                        (dogController.list.filter(dog => dog.type === DogType.beagle).length !==0
                            || dogController.list.filter(dog => dog.type === DogType.yorkshireTerrier).length !== 0
                            || dogController.list.filter(dog => dog.type === DogType.chihuaua).length !== 0
                            || dogController.list.filter(dog => dog.type === DogType.poodle).length !== 0
                            || dogController.list.filter(dog => dog.age <= 12).length !== 0)"
            dogs="dogController.list | littleDogs">
    </dogbox>
. . .
</html>
```

Einführen einer neuen Property showLittleDogList im ViewModel

```
public get showLittleDogList(){
     return this.list.filter(dog => dog.type === DogType.beagle).length !==0
                              || list.filter(dog => dog.type === DogType.yorkshireTerrier).length !== 0
                              || list.filter(dog => dog.type === DogType.chihuaua).length !== 0
                              || list.filter(dog => dog.type === DogType.poodle).length !== 0
                              || list.filter(dog => dog.age <= 12).length !== 0);
}
```

```
<html>
. . .
    <dogbox id="boxOfLittleDogs"
            is-visible="dogController.showLittleDogList"
            dogs="dogController.list | littleDogs">
    </dogbox>
. . .
</html>
```

```
public get showLittleDogList(){
     return this.list.filter(dog => dog.type === DogType.beagle).length !==0
                              || list.filter(dog => dog.type === DogType.yorkshireTerrier).length !== 0
                              || list.filter(dog => dog.type === DogType.chihuaua).length !== 0
                              || list.filter(dog => dog.type === DogType.poodle).length !== 0
                              || list.filter(dog => dog.age <= 12).length !== 0);
}
```

=> Testbar mittels Unit-Tests der Property

Diese Property kann nun ganz einfach durch überschaubares Refactoring in einen akzteptablen Zustand gebracht werden (z.B. Einführen einer Liste wie in vorherigen Bsp.)

```
protected createDogListRenderer() {
            return new GridColumn("Name", "Description", (data, type, full: Dog) => {
                let creationDate: string = this.$filter('date')(full.creationTime, 'medium');
                let dogItemStyle = "";
                if (full.character !== null) {
                    dogItemStyle += "border-color: " + this.getHighlightColour(full.character) + ";";
                }

                return
                    `<div class="dogItem" style="` + dogItemStyle + `">` +
                      `<div class="firstRow">` +
                       `<svg-icon data-size="24"> ={{"` + full.icon + `"}} </svg-icon>` +
                      `</div>` +
                      `<div class="secondRow">` +
                         `<span>` + full.type + `</span>` +
                         `<span>` + full.name + `</span>` +
                       `</div>` +
                       `<div class=„thirdRow">` +
                            full.birthDate +
                       `</div>` +
                    `</div>`
            }
}
```

Diesen Teil in eine Direktive ziehen

```
interface ILittleDogDirectiveScope extends ng.IScope {
    dog: IDog;
 }

class LittleDogViewModel {
    constructor(private $scope: ILittleDogDirectiveScope){}

    public get type() {
        return this.$scope.dog.type;
    }
    . . .
}
angular.module('MyAngularModule')
    .controller('littleDogViewModel‘, LittleDogViewModel);

angular.module(„MyAngularModule")
    .directive("littleDog", (): ng.IDirective => {
        return {
            scope: {
                dog: '='
            },
            controller: 'littleDogDirective',
            controllerAs: 'littleDog',
            bindToController: true,
            templateUrl: 'apps/shared/view/littleDog.tpl.html'
        }
    });
```


```
interface ILittleDogDirectiveScope extends ng.IScope {
    dog: IDog;
 }

class LittleDogViewModel {
    constructor(private $scope: ILittleDogDirectiveScope){}

    public get type() {
        return this.$scope.dog.type;
    }
    . . .
}
angular.module('MyAngularModule')
    .controller('littleDogViewModel', LittleDogViewModel);

angular.module(„MyAngularModule")
    .directive("littleDog", (): ng.IDirective => {
        return {
            scope: {
                dog: '='
            },
            controller: 'littleDogDirective',
            controllerAs: 'littleDog',
            bindToController: true,
            templateUrl: 'apps/shared/view/littleDog.tpl.html'
        }
    });
```

Unit-Tests auf Property-Ebene für ViewModel

```
interface ILittleDogDirectiveScope extends ng.IScope {
    dog: IDog;
 }

class LittleDogViewModel {
    constructor(private $scope: ILittleDogDirectiveScope){}

    public get type() {
        return this.$scope.dog.type;
    }
    . . .
}
angular.module('MyAngularModule')
    .controller('littleDogViewModel‘, LittleDogViewModel);
```

```
angular.module(„MyAngularModule")
    .directive("littleDog", (): ng.IDirective => {
        return {
            scope: {
                dog: '='
            },
            controller: 'littleDogDirective',
            controllerAs: 'littleDog',
            bindToController: true,
            templateUrl: 'apps/shared/view/littleDog.tpl.html'
        }
    });
```

Wenn notwendig, bieten Testframeworks Mittel zum Testen von Direktiven

```
<html>
   <div class="dogItem"> <!-- Style moved to css-->
        <div class="firstRow">
            <svg-icon size="24"> {{littleDog.icon}} </svg-icon>
        </div>
        <div class="secondRow">
            <span> {{littleDog.type}} </span>
            <span> {{littleDog.name}} </span>
        </div>
        <div class="thirdRow">
            <time> {{littleDog.birthDate}} </time>
        </div>
    </div>
</html>
```

- Reine View
- Style kann in css verschoben werden
- Übersichtlichere Formatierung

Aufruf:

```
<html>
. . .
   <little-dog dog="listController.dogToBeShown"> </little-dog>
. . .
</html>
```

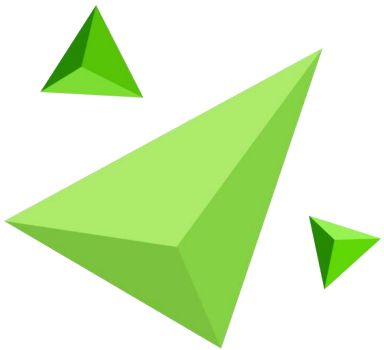

Beispiel

OO-Prinzipien verletzt

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    let start: number = data[3].value || 0;
    let end: number =  data[4].value;

    this.searchParameters.startIndex = start;
    this.searchParameters.endIndex = end;
    this.searchParameters.filterExpression = '';
    this.userService.loggedInUserPromise
        .then(() => this.dogManagementRestClient.getColliRange(this.searchParameters))
        .then((result: ResultBase<Dog>) => {
            var dogList = <ILoadData<Dog>><any>result.records.slice();
            dogList.sEcho = data[0].value;
            dogList.iTotalDisplayRecords = result.totalDisplayRecords;
            dogList.iTotalRecords = result.totalRecords;

            callback(dogList);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                // this message is shown already on open dialog
                callback(<ILoadData<Dog>><any>{});
                return;
            }
            this.errorHandler.handleWithDialog(response);
        });
}
```

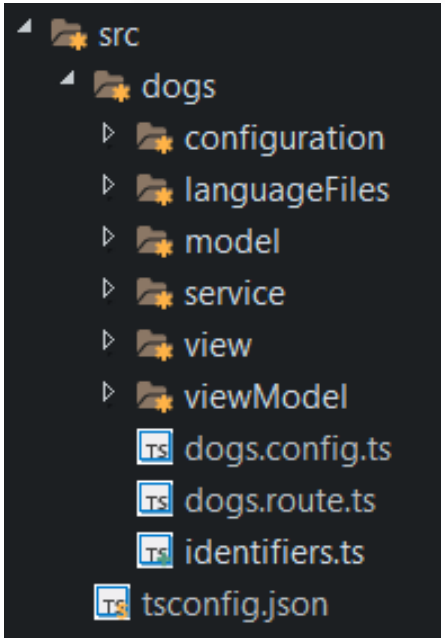

```
public loadDataList = (notneeded1, data, callback, notneeded2): void => {
    this.loadDataFromServer(new DataPage(data))
        .then((result: ILoadData<Dog>) => {
            callback(result);
        }).catch((result: ILoadData<Dog>) => {
            callback(result);
        });
}
public loadDataFromServer = (page: DataPage): ng.IPromise<ILoadData<Dog>> => {
    this.searchParameters = new SearchParameters(page);
    return this.userService.loggedInUserPromise
        .then(() => {
            return this.getDogsFromServer(page.sEcho);
        })
        .catch((response: HttpResponseException) => {
            if (response.data.message === "NotAuthenticated") {
                return new LoadData<Dog>(page.sEcho);
            }
            this.errorHandler.handleWithDialog(response);
        });
 }
 private getDogsFromServer = (sEcho: number): ng.IPromise<ILoadData<Dog>> => {
    return this.dogManagementRestClient.getColliRange(this.searchParameters)
        .then((result: ResultBase<Dog>) => {
            return new LoadData<Dog>(sEcho, result);
        });
 }
```


Fazit



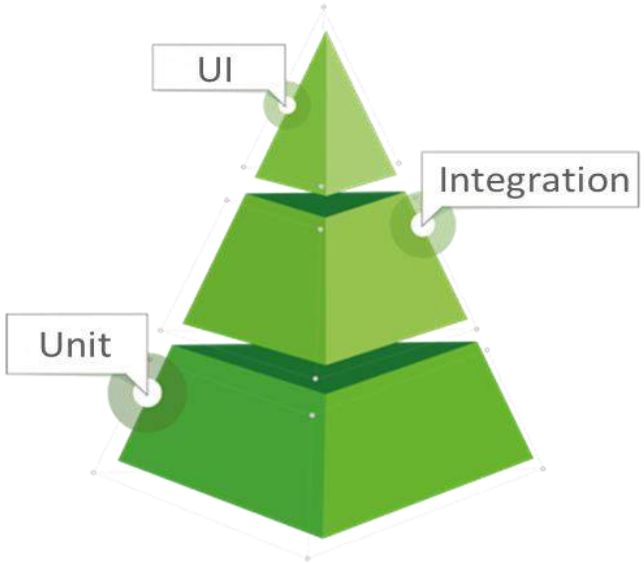
Überblick neue Struktur

- Allgemein verwendete Funktionalitäten in Bibliothek verschoben (eigenes Projekt)
- service: angular-Services (für App benötigte Funktionalitäten)
- view: HTML-Templates
- viewModel: angular-Controller zu den Views
- model: angular-Service → **kein $scope inject mehr nötig**



Fazit

- Keine unnötigen Abhängigkeiten mehr vorhanden
  - Tests schreiben ist einfacher
- Code Qualität hat sich verbessert
  - Lesbarkeit
  - Erweiterbarkeit
  - Wartbarkeit
- Testabdeckung UnitTest: 75% (alt 20%)
  - Nur wenige UI-Tests nötig



Quellen

- https://www.andrena.de/files/andrena/media/dokumente/fachartikel/2016/sonderdruck_knapp_yilmaz_OS_02_16_web.pdf
- https://martinfowler.com/articles/practical-test-pyramid.html
- https://martinfowler.com/articles/mocksArentStubs.html
- Object-Oriented Software Construction; Bertand Meyer,1997
- Clean Code; Robert C. Martin, 2008
- Wikipedia