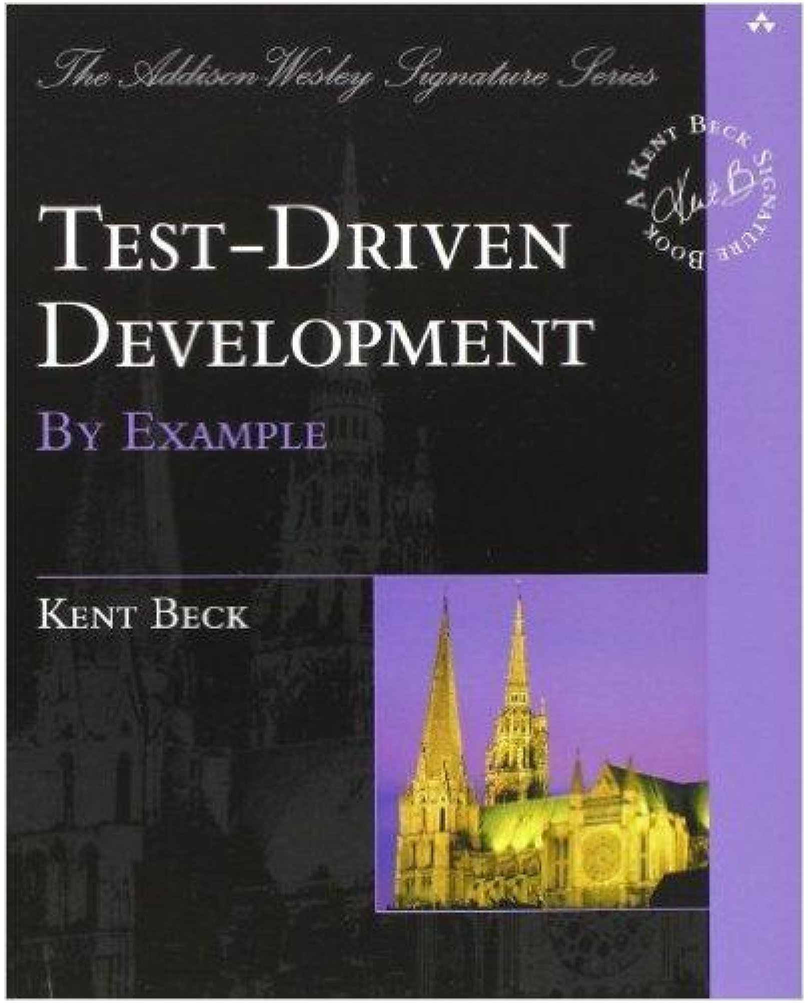
TDD for dummies

“I'm not a great programmer; I'm just a good programmer with great habits.”
– **Kent Beck**


The Addison Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

Where do I start?

FIRST SOLID!

Testing

1. **F**ast
2. **I**ndependent
3. **R**epeatable
4. **S**elf-checking
5. **T**imely

Object Oriented Design

1. **S**ingle responsability
2. **O**pen/closed principle
3. **L**iskov substitution principle
4. **I**nterface segregation principle
5. **D**ependency inversion principle

S.O.L.I.D.

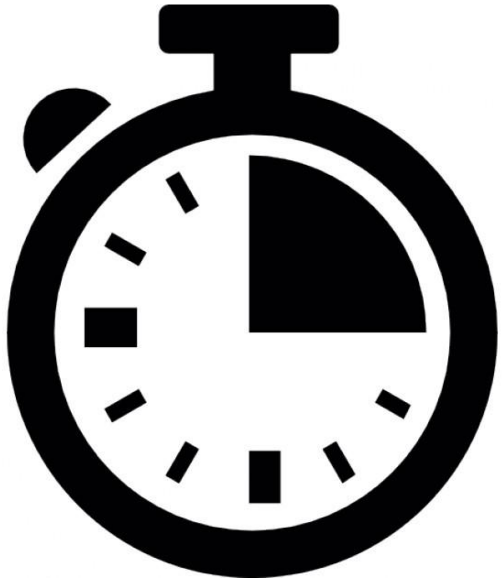
- **Single responsibility**: a class should have only one reason to change.
- **Open Close Principle**: open for extension but closed for modifications.
- **Liskov's Substitution Principle**: derived types must be completely substitutable for their base types.
- **Interface Segregation Principle**: Clients should not be forced to depend upon interfaces that they don't use.
- **Dependency Inversion Principle**:
  - High-level modules should not depend on low-level modules. Both should depend on abstractions.
  - Abstractions should not depend on details. Details should depend on abstractions.

F.I.R.S.T.

FAST

A developer should not hesitate to run the tests as they are slow.

Is an asynchronous test fast enough?



F.**I.**R.S.T.

Isolated/Independent

No tests depend on others, so can run any subset in any order.

What about fixtures in a Database?

F.I.**R.**S.T.

Repeatable

Run N times, get same result.

Is a test depending upon a remote service, repeatable?

F.I.R.**S.**T.

Self-validating

Test can automatically detect if passed. No manual inspection required.

Is *print n_passed* + *" tests passed"* self validating?

F.I.R.S.**T.**

Timely

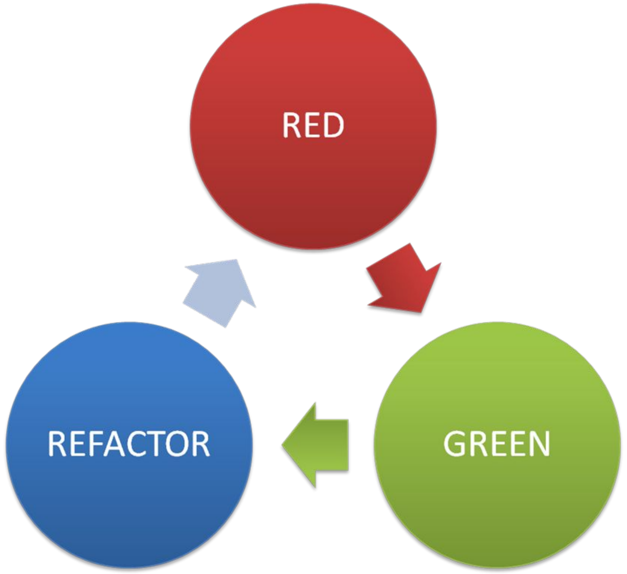
Elaborated and written before the code.

Is the following test timely elaborated?

expect(MyClass).not.toBe(undefined);

Red Green Refactor

1. Red: write a little test that doesn't work, perhaps doesn't even compile at first.
2. Green: make the test work quickly, committing whatever sins necessary in the process.
3. Refactor: eliminate all the duplication created in just getting the test to work.



Test to-do list

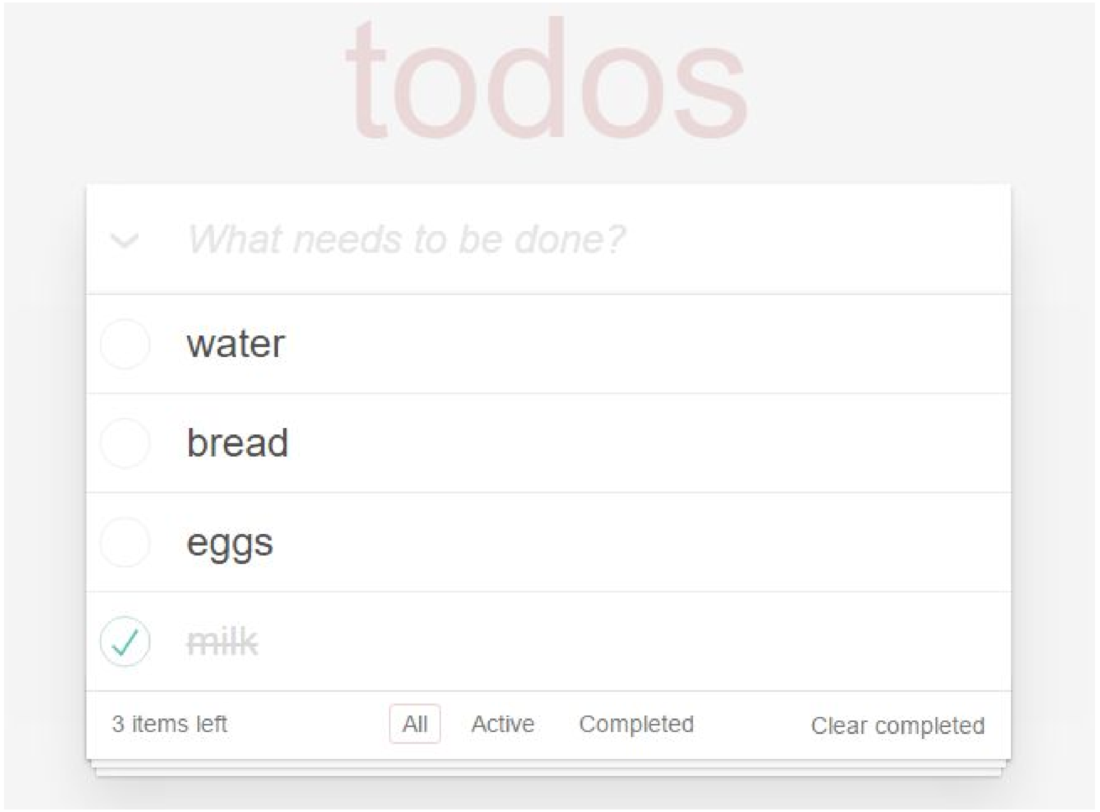
- Remind us what all we need to do, keep us focused
- When we start working on an item, we'll make it **bold**
- When we finish an item we'll ~~cross it off~~
- When we think of another test to write, we'll add it
- When we have a doubt, we write it down on the list

Let’s begin our Red-Green-Refactor cycle on an example.

To Do application

Write a web application that lets you:

- Note down things to do
- Mark things as done
- Remove noted down items
- Change items description


todos

What needs to be done?

water

bread

eggs

~~milk~~

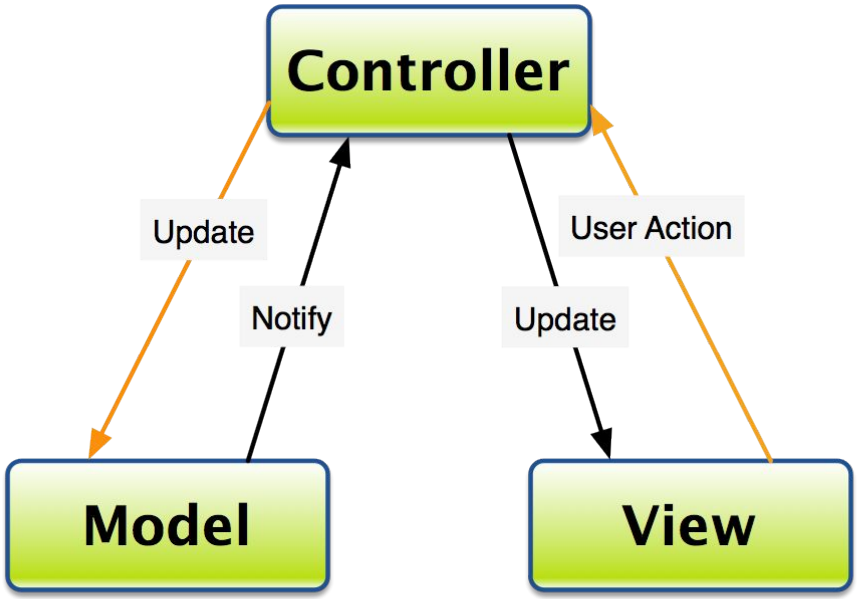
3 items left

ToDo app: a bit of analysis

Where do we start?

Clearly the main pattern here is MVC and the requirements are a typical C.R.U.D.

We can start testing the logic that will show a list of todos.



ToDo app: before we start

When coding a web application, we must decide which framework we are going to use. So the best testing framework is related to which framework we choose.

To keep this example simple we will:

- Not consider the view
- Use pure JavaScript (so no framework)
- Use Jasmine testing framework

Initial setup

```
describe('ToDoCtrl', function() {
 var testView;

 beforeEach(function() {
   testView = {
     update: function() {}
   };

   spyOn(testView, 'update');
 });

});
```

Test list

- **Show a list of todo**
- Note down things to do
- Mark things as done
- Remove noted down items
- Change items description

First test: Red

```
beforeEach(...);

it('should show a list of Todos', function() {
 // Arrange
 var model = [{text: 'ToDo 1'}, {text: 'ToDo 2'}];
 var ctrl = new ToDoCtrl(model);

 // Act
 ctrl.init(testView);

 // Assert
 expect(testView.update).toHaveBeenCalledWith(model);
});
```

Test list

- **Show a list of todo**
- Note down things to do
- Mark things as done
- Remove noted down items
- Change items description

First test: Green

```
'use strict';

function ToDoCtrl(model) {
   this._model = model;
}

ToDoCtrl.prototype.init = function(view) {
   view.update(this._model);
};
```

Test list

- **Show a list of todo**
- Note down things to do
- Mark things as done
- Remove noted down items
- Change items description

Doubts:

- Is the model OK?
- Shouldn’t be there a service to retrieve the model?

onDone handler: Red

```
it('should register a onDone handler', function() {
 var ctrl = new ToDoCtrl([]);
 spyOn(testView, 'onDone').and.callThrough();

 ctrl.init(testView);

 expect(testView.onDone)
        .toHaveBeenCalledWith(testView._doneHandler);
});
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
    - **Controller should register an onDone handler**
    - Controller should be able to mark a todo as done
    - Controller should be able to mark a todo as not done
- ~~Show a list of todos~~

Doubts:

- Shouldn’t be there a service to retrieve the model?

onDone handler: Green

```
'use strict';

function ToDoCtrl(model) {
 this._model = model;
}

ToDoCtrl.prototype.init = function(view) {
 view.onDone(function() {});
 view.update(this._model);
};
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
  - **Controller should register an onDone handler**
  - Controller should be able to mark a todo as done
  - Controller should be able to mark a todo as not done
- ~~Show a list of todos~~

Doubts:

- Shouldn’t be there a service to retrieve the model?

onDone handler: Refactor

```
'beforeEach(function() {
  testView = {...};

  ~~spyOn(testView, 'update');~~
});

it('should show a list of Todos', function () {
 var model = [{text: 'ToDo 1'}, {text: 'ToDo 2'}];
 var ctrl = new ToDoCtrl(model);

 spyOn(testView, 'update');

...
});
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
  - ~~**Controller should register an onDone handler**~~
  - Controller should be able to mark a todo as done
  - Controller should be able to mark a todo as not done
- ~~Show a list of todos~~

Doubts:

- Shouldn’t be there a service to retrieve the model?

Mark as done: Red

```
it('should mark a todo as done when the user taps on it', function () {
 var model = [{text: 'A todo', done: false}];
 var ctrl = new ToDoCtrl(model);
 ctrl.init(testView);

 var todoIndex = 0;
 testView._doneHandler(todoIndex);

 expect(model[todoIndex]).toEqual({text: 'A todo', done: true});
});
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
    - ~~Controller should register an onDone handler~~
    - **Controller should be able to mark a todo as done**
    - Controller should be able to mark a todo as not done
- ~~Show a list of todos~~

Doubts:

- Shouldn’t be there a service to retrieve the model?

Mark as done: Green

```
function ToDoCtrl(model) {
 this._model = model;
}

ToDoCtrl.prototype.init = function(view) {
 var self = this;
 view.onDone(function(todoIndex) {
   self._model[todoIndex].done = true;
 });
 view.update(this._model);
};
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
    - ~~Controller should register an onDone handler~~
    - ~~**Controller should be able to mark a todo as done**~~
    - Controller should be able to mark a todo as not done
- ~~Show a list of todos~~

Doubts:

- Shouldn’t be there a service to retrieve the model?

Mark as undone: Red

```
it('should mark a todo as not done when the user taps on a done
todo', function () {

 var model = [{text: 'A todo', done: true}];
 var ctrl = new ToDoCtrl(model);
 ctrl.init(testView);

 var todoIndex = 0;
 testView._doneHandler(todoIndex);

 expect(model[todoIndex]).toEqual({text: 'A todo', done: false});
});
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
  - ~~Controller should register an onDone handler~~
  - ~~Controller should be able to mark a todo as done~~
  - **Controller should be able to mark a todo as not done**
- ~~Show a list of todos~~

Doubts:

- Shouldn't be there a service to retrieve the model?

Mark as undone: Green

```
function ToDoCtrl(model) {
 this._model = model;
}

ToDoCtrl.prototype.init = function(view) {
 var self = this;
 view.onDone(function(todoIndex) {
   self._model[todoIndex].done = !self._model[todoIndex].done;
 });
 view.update(this._model);
};
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- Mark things as done
  - ~~Controller should register an onDone handler~~
  - ~~Controller should be able to mark a todo as done~~
  - **Controller should be able to mark a todo as not done**
- ~~Show a list of todos~~

Doubts:

- Shouldn't be there a service to retrieve the model?

Mark as undone: Refactor

```
it('should mark a todo as done when the user taps on it', function () {
 var model = [{text: 'A todo', done: false}];
 checkDoneChangedOnTap(model);
});

it('should mark a todo as not done when the user taps on a done todo', function () {

 var model = [{text: 'A todo', done: true}];
 checkDoneChangedOnTap(model);
});
```

Test list

- Note down things to do
- Remove noted down items
- Change items description
- ~~Mark things as done~~
  - ~~Controller should register an onDone handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~**Controller should be able to mark a todo as not done**~~
- ~~Show a list of todos~~

Doubts:

- Shouldn't be there a service to retrieve the model?

ToDoService: Introduce a test ToDoService

```
function TestTodoService(todos) {
 this._todos = todos;
}

TestTodoService.prototype.todos = function() {
 return this._todos;
};


describe('ToDoCtrl', function() {

...

});
```

Test list

- ...
- **Shouldn’t be there a service to retrieve the model? Yes**
  - **Introduce a TestTodoService**
  - Update tests to use TestTodoService
  - Remove production code dependency from TestTodoService
- Mark things as done
  - controller should save the changes when a todo is changed
  - ~~Controller should register an onDone handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~
- ~~Show a list of todos~~

ToDoService: Introduce a test ToDoService

```
function ToDoCtrl(todoService) {
 if (Array.isArray(todoService)) {
   this._service = new TestTodoService(todoService);
 } else {
   this._service = todoService;
 }

 this._model = this._service.todos();
}

ToDoCtrl.prototype.init = function(view) {
          ...
};
```

Test list

- ...
- **Shouldn't be there a service to retrieve the model? Yes**
  - **Introduce a TestTodoService**
  - Update tests to use TestTodoService
  - Remove production code dependency from TestTodoService
- Mark things as done
  - controller should save the changes when a todo is changed
  - ~~Controller should register an onDon handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~
- ~~Show a list of todos~~

ToDoService: Update tests

```
it('should show a list of Todos', function() {
 var model = [{text: 'ToDo 1'}, {text: 'ToDo 2'}];

 var ctrl = new ToDoCtrl(new TestTodoService(model));
 spyOn(testView, 'update');

 ctrl.init(testView);

 expect(testView.update).toHaveBeenCalledWith(model);
});

it('should register a onDone handler', function() {
          ...
});

...
```

Test list

- ...
- **Shouldn't be there a service to retrieve the model? Yes**
  - ~~Introduce a TestTodoService~~
  - **Update tests to use TestTodoService**
  - Remove production code dependency from TestTodoService
- Mark things as done
  - controller should save the changes when a todo is changed
  - ~~Controller should register an onDor handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~

ToDoService: Remove test dependecy

```
function ToDoCtrl(todoService) {
 this._service = todoService;
 this._model = this._service.todos();
}

ToDoCtrl.prototype.init = function(view) {
 var self = this;
 view.onDone(function(todoIndex) {
   self._model[todoIndex].done = !self._model[todoIndex].done;
 });
 view.update(this._model);
};
```

Test list

- ...
- **Shouldn't be there a service to retrieve the model? Yes**
  - ~~Introduce a TestTodoService~~
  - ~~Update tests to use TestTodoService~~
  - **Remove production code dependency from TestTodoService**
- Mark things as done
  - controller should save the changes when a todo is changed
  - ~~Controller should register an onDor handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~

Save changes on tap: Red

```
it('should save the model when a todo is changed', function() {
 var model = [{
   text: 'A todo',
   done: true
 }];
 var service = new TestTodoService(model);
 spyOn(service, 'save');
 new ToDoCtrl(service).init(testView);

 testView._doneHandler(0);

 expect(service.save).toHaveBeenCalled();
});
```

Test list

- ...
- **Shouldn't be there a service to retrieve the model? Yes**
  - ~~Introduce a TestTodoService~~
  - ~~Update tests to use TestTodoService~~
  - ~~Remove production code dependency from TestTodoService~~
- Mark things as done
  - **controller should save the changes when a todo is changed**
  - ~~Controller should register an onDor handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~

Save changes on tap: Green

```
function ToDoCtrl(todoService) {
 this._service = todoService;
 this._model = this._service.todos();
}

ToDoCtrl.prototype.init = function(view) {
 var self = this;
 view.onDone(function(todoIndex) {
   self._model[todoIndex].done = !self._model[todoIndex].done;
   self._service.save();
 });
 view.update(this._model);
};
```

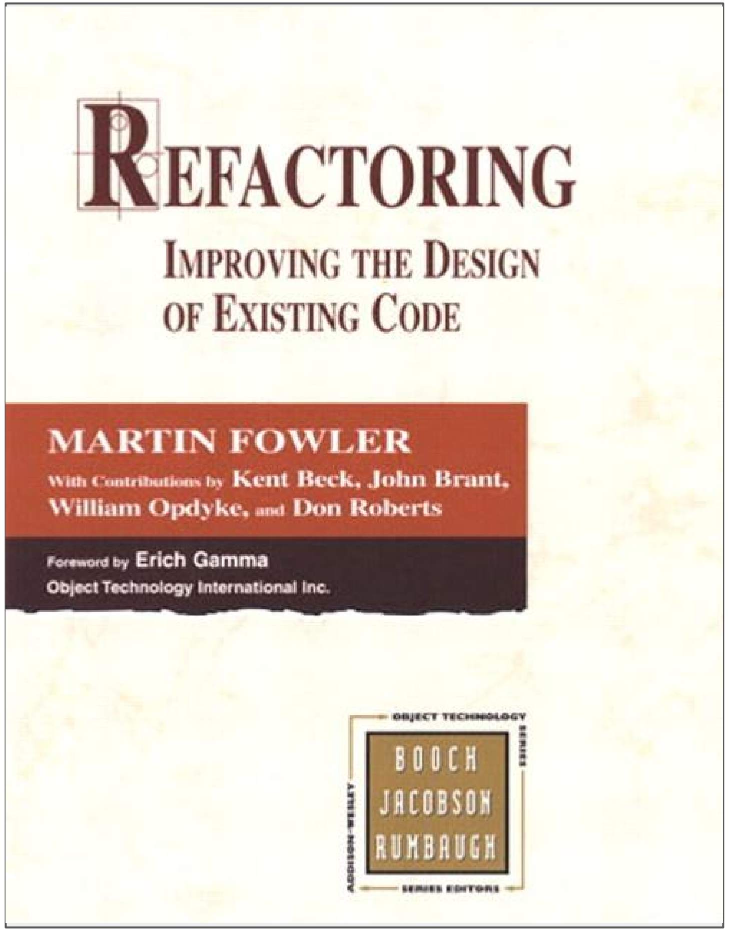
Test list

- ...
- **Shouldn't be there a service to retrieve the model? Yes**
  - ~~Introduce a TestTodoService~~
  - ~~Update tests to use TestTodoService~~
  - ~~Remove production code dependency from TestTodoService~~
- Mark things as done
  - **controller should save the changes when a todo is changed**
  - ~~Controller should register an onDor handler~~
  - ~~Controller should be able to mark a todo as done~~
  - ~~Controller should be able to mark a todo as not done~~

Test list

- Note down things to do
- Remove noted down items
- Change items description
- **Mark things as done**
    - ~~Controller should save the changes when a todo is changed~~
    - ~~Controller should register an onDone handler~~
    - ~~Controller should be able to mark a todo as done~~
    - ~~Controller should be able to mark a todo as not done~~
    - **Controller should update the view when a todo is changed**
- ~~Show a list of todos~~
- ~~Introduce a test TodoService~~
- Create a TodoService that save into localStorage

Legacy code?


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY