UI Testing with Spec

The future is here… hace rato!

Pablo Tesone
Pharo Consortium Engineer
If it has no tests…

it does not exist.

Dr. Test (1987 - ...)
A little strong… but…

- Missing Tests
  - Fear of Changes
  - Unknown Impact
  - Bad Surprises
  - Pain… lots of pain…
Really…
If I delete something or break it…

How long it will take to detect the error?

We all love tests. That is easy.
We need to test the UI

Testing UI is difficult
We need special tools
Selenium, Watir, Cypress, or Cucumber
Es al pedo!
j'ai la flemme!

with just Objects & Polymorphism
2 similar but different problems.

- **Testing Spec implementation itself** (Adapters, Presenters, Widgets, Layouts, Backends, etc)

- **Testing Applications written in Spec** (display, interactions, update, navigations)

Before Refactoring... we need tests!
Testing Spec

- Spec is a big monster, maybe not so big... but scary... maybe not so scary:

- Spec has a nice modular implementation, different objects with different responsibilities
Testing Spec

- Presenters
  - Interaction with the Model
  - Events
  - Public API
  - Default Values

- Adapters
  - Interaction Presenters / Widget
  - Creating Widgets
  - Events
  - Same Behaviour in each backend

- Widgets
  - Backend API
  - Widgets themselves
  - Events

- Layouts
  - How to create widgets
  - Where to put them

Lots of Small tests!!

Stop Complaining, there are not so many.
Common Scenarios

When Test Is Executed

Before Opening
- Modify Presenter
- Open Widget
- Asserts

After Opening
- Open Widget
- Modify Presenter
- Asserts

Different Backends
- GTK
- Morphic
- Other

We want it for all tests
**Testing List Adapter:** When I select something in the presenter it is propagated to the widget

```small
(testSelectPresenterIndexSetsSelectedIndexInWidget
  presenter selectIndex: 1.
  self assert: (self widget selectedIndexes includes: 1))
```

- Only 1 Simple Test Case
- Simple / Multiple Selection
- With / Without Columns
- Widget Created / Not Created
- Gtk / Morphic

All this for 13 different selection scenarios

Only List and Selection: 208 Tests
Proposed Solution: Coding Monkeys

Just Kidding, we are lazy... you should be lazy also.
Implementing it with “Style”

Matrix of Parameters

Parametrized Tests

Generates when you run the suite.

A Test Instance for each combination of parameters
Our Matrix

AbstractAdapterTest class >> #testParameters

^ ParametrizedTestMatrix new

When

  forSelector: #specInitializationStrategy
  addOptions: { [ SpecOpenStrategy openBefore ].
    [ SpecOpenStrategy openAfter ] };

  forSelector: #backendForTest
  addOptions: AbstractBackendForTest allSubclasses;

yourself

Backend
We want simple tests!

```smalltalk

testSelectItemSelectsTheGivenElement

    self presenter selection selectedPath: #(2).
    self assert: self adapter selectedItem equals: 2.


testSettingAnImageSetsTheImage

    self presenter image: self imageForm.
    backendForTest assertImage: self adapter image equals: self imageForm.
```
Something else required...

- Putting in the test backend depending code
  
  **Example:**
  Asserting if two images are the same
  
  `#assertImage:equalsForm:`

  Clicking / Selecting of a widget / etc.

- Adding Testing methods to the adapters & presenters
  
  **Example:**
  - Emulating Events.
  - Getting State
  - Accessing real widget
Results

- Lots of Tests: 1400+ and growing
- Easy To develop new ones / Easy to maintain.
- Validation of Public API
- Validation of Backend API => Easy to implement new Backends.
Second Problem: Testing Applications

• Easy, let’s create Tests.

• In Spec we believe, let’s test the application

Maybe Spec has problems. But let’s create tests where they should be.
Example Application

Phonebook

PhoneEntry
name
company
mail
telephone

initializeWidgets
entriesList := self newList
    whenSelectionChangedDo: [:sel |
        detailsPanel model: sel selectedItem.
        removeButton enabled: sel isEmpty not ];
    yourself.

addButton := self newButton
    label: 'Add';
    yourself.

removeButton := self newButton
    label: 'Remove';
    action: [ self removeEntry ]
    yourself.

detailsPanel := self
    instantiate: PhonebookEntryPresenter
    on: nil.

Pharo Phonebook

Phonebook

Phonebook Entry Presenter

PhonEntry

Buttons

Details panel
Testing Widgets

• Testing that a widget is shown

  testWindowHasAddButton

    self assert: (window hasPresenter:
                  presenter addButton)

• Testing that a widget is correctly initialised

  testAddButtonHasLabel

    self assert: presenter addButton label equals: 'Add'

Seems stupid, but we can test i18N here!
Testing UI State Update

• Selecting an element update the UI

```smalltalk
testSelectingAnElementEnablesRemoveButton

presenter entriesList selectIndex: 1.
self assert: presenter removeButton isEnabled
```

```smalltalk

testSelectingAnElementUpdatesDetailName

presenter entriesList selectIndex: 1.
self assert: presenter detailsPanel nameLabel label equals: 'A Person'.
```
Testing UI Interactions

- Clicking in Remove

```smalltalk
testClickingRemoveButtonRemovesDisablesRemoveButton

presenter entriesList selectIndex: 1.
presenter removeButton click.

self deny: presenter removeButton isEnabled
```

```smalltalk
testClickingRemoveButtonRemovesAnElementFromTheList

presenter entriesList selectIndex: 1.
presenter removeButton click.

self assert: presenter entriesList items size equals: 0
```
Testing UI Layout

testAddButtonIsBelowEntryList

self assert: (presenter addButton isBelowOf: presenter entriesList)

testAddButtonIsLeftOfRemoveButton

self assert: (presenter addButton isLeftOf: presenter removeButton)

Also we are able to test dynamic things!
• Testing Navigation

```small

testClickingAddButtonOpenANewWindow

  presenter addButton click.
  self assert: presenter application windows size equals: 2

  testClickingAddButtonOpenCorrectWindow

  presenter addButton click.
  self assert: presenter application focusedPresenter class
  equals: PhonebookAddEntryPresenter
```

*Once open... it is responsibility of other test to test it*
Testing UI

- Spec Applications are easily testable.
- Centring on relation between our presenters.
- Spec provides methods for testing.
Thanks!

• Adding Testing infrastructure to Spec2.
• Testing implementation and backends.
• Expressing the contracts with backend as tests.
• Open to new backend implementations.
• Support for Application Testing.
• Writing UI Tests as another easy test.

Now... without excuses.

May the tests be with you!