How to test a Spec App ?

Christophe Demarey
Inria Engineer

with the support of Esteban Lorenzano
and Pablo Tesone

Why would you need to test your GUI?

GUI => primary way users interact with your app

Something broken => user will feel the whole app is broken even if the underlying functionality is sound.

Ensure functionality

- Verifies that buttons, menus, forms, and other elements work as expected

- Ensures that user actions trigger the correct events and responses

Prevents Regressions

- Automated GUI tests can detect if a change in one part of the interface unintentionally breaks another.

- Ensures continuity in the look and behavior of the UI after updates.

UI test good practice

Use an instance var to hold window opened during test

```
TestCase << #SptImageAnnotationPresenterTest
    slots: { #window };
    tag: 'Presenters';
    package: 'Spec2-Tutorial-Tests'
```

Define a tearDown method to close opened window

```
tearDown

    window ifNotNil: [ window close ].
    super tearDown
```

UI test good practice - alternative when many windows

Use an instance var to hold the presenter under test

```
TestCase << #SptImageMetadataPresenterTest
    slots: { #presenter };
    tag: 'Presenters';
    package: 'Spec2-Tutorial-Tests'
```

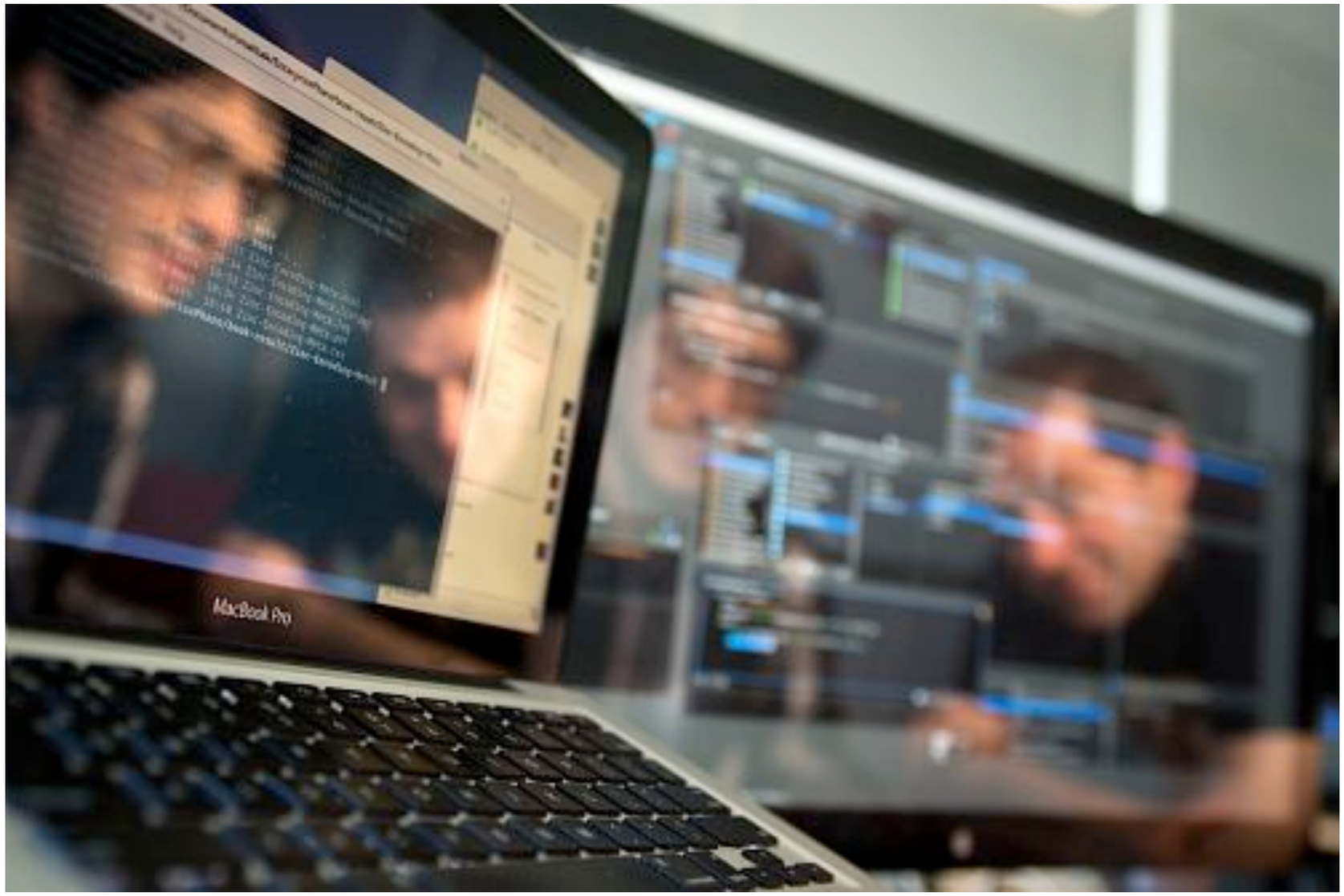
Define a tearDown method to close all opened windows

```
tearDown

    presenter ifNotNil: [
        presenter application windows
            do: [ :window | window close ] ].

    super tearDown
```

What could we test?

- Test widgets
- Test UI state update
- Test UI interaction
- (Test UI layout)

Now ~~coding~~ testing!



© Inria / Photo C. Morel

Test Window title

```
testPresenterWindowTitleIsCorrect

    | presenter |
    presenter := SptImageAnnotationPresenter new.

    window := presenter open.

    self assert: window title equals: 'Image Annotation app'
```

Structuring test with AAA pattern

testPresenterWindowTitleIsCorrect

Arrange

```
| presenter |
presenter := SptImageAnnotationPresenter new.
```

Act

```
window := presenter open.
```

Assert

```
self
    assert: window title
    equals: 'Image Annotation app'
```

Test widget

```
testTag1LabelIsCorrect
    | imagePresenter |
    imagePresenter := SptImageMetadataPresenter on: nil.

    self assert: imagePresenter tag1Button label equals: 'expo 1'
```

Test opened windows

```
testClickingOnEditMetadataOpenTwoWindows

    imagePresenter := SptImageMetadataPresenter
                        on: SptTestImageMetadata example.

imagePresenter open.
imagePresenter clickOnUpdateMetadata.

self assert: imagePresenter application windows size equals: 2
```

Test UI state update

```
testResetDescButtonIsDisabledWhenNoImageDescription

    imagePresenter := SptImageMetadataPresenter on:
                          SptTestImageMetadata example.

    self deny: imagePresenter resetDescButton isEnabled
```

```
testResetDescButtonIsEnabledAfterImageDescriptionSet

    imagePresenter := SptImageMetadataPresenter on:
                          SptTestImageMetadata example.
    self deny: imagePresenter resetDescButton isEnabled.

    imagePresenter clickOnTag1.

    self assert: imagePresenter resetDescButton isEnabled
```

Test interaction

Selecting an item in a list

should update another presenter

testSelectingAKeyUpdateValueFieldAccordingly

```
presenter := SptMetadataEditorPresenter on: SptImageMetadata example.

presenter keyList selectFirst.

self assert: presenter valueInput text equals: 'jpg'
```

Fill inputs
and check behaviour

testCanUpdateSelectedMetadata

```
    | model newValue |
    model := SptTestImageMetadata example.
    presenter := SptMetadataEditorPresenter on: model.
    presenter keyList selectItem: #Make.
    newValue := 'my new data'.

    presenter valueInput text: newValue.
    presenter clickOnUpdate.

    self assert: (model metadataNamed: #Make) equals: newValue
```

Test when window is modal

```
testClickingOnResetOpenTwoWindows

    imagePresenter := SptImageMetadataPresenter
                          on: SptTestImageMetadata example.

    imagePresenter open.
    SpWindowForceOpenNonModal
        during: [ imagePresenter clickOnResetImageDescription ].

    self assert: imagePresenter application windows size equals: 2
```

Test when window is modal

```
SpWindowForceOpenNonModal during: aBlock
```

Will force the opening of windows as non-modal windows during the execution of aBlock.

Execution is not blocked anymore !

Testing modals

Ok, but … how do I test if I need to interact with the presenter opened as modal ?

```
SpWindowSimulateOpenModal
        value: [ :aDialogPresenter | ]
        during: aBlock.
```

Will open a modal presenter as non-modal and give you back the opened presenter so that you can interact with it:

Set a value, choose an option, click on a button.

Test with confirm dialog

```
testResetingImageDescriptionMetadataShouldRemoveItFromImageDescriptionInImage

    imagePresenter := SptImageMetadataPresenter
                        on: (SptTestImageMetadata example
                                imageDescription: 'testResetingImageDescription';
                                yourself).
    imagePresenter model: imagePresenter model.
    self
        assert: (imagePresenter metadataNamed: 'ImageDescription')
        equals: 'testResetingImageDescription'.

    SpWindowSimulateOpenModal
        value: [ :aDialogPresenter | aDialogPresenter triggerOkAction ]
        during: [ imagePresenter clickOnResetImageDescription ].

    self
        should: [ imagePresenter metadataNamed: 'ImageDescription' ]
        raise: NotFound
```

Thanks!

pharo-spec/spec-tuto

Will come soon: step by step instructions for the tutorial