Lightweight **Unit Testing** for **Titanium**

Yakup Kalin
@yakupkalin

David Cypers
@davidcyp

A.C.A MOBILE

A.C.A IT-Solutions

“

Unit testing? **We don’t**.
We always TEST the whole app in the container.

FokkeZB, Appcelerator Developer Evangelist, **April 6, 2016**

”

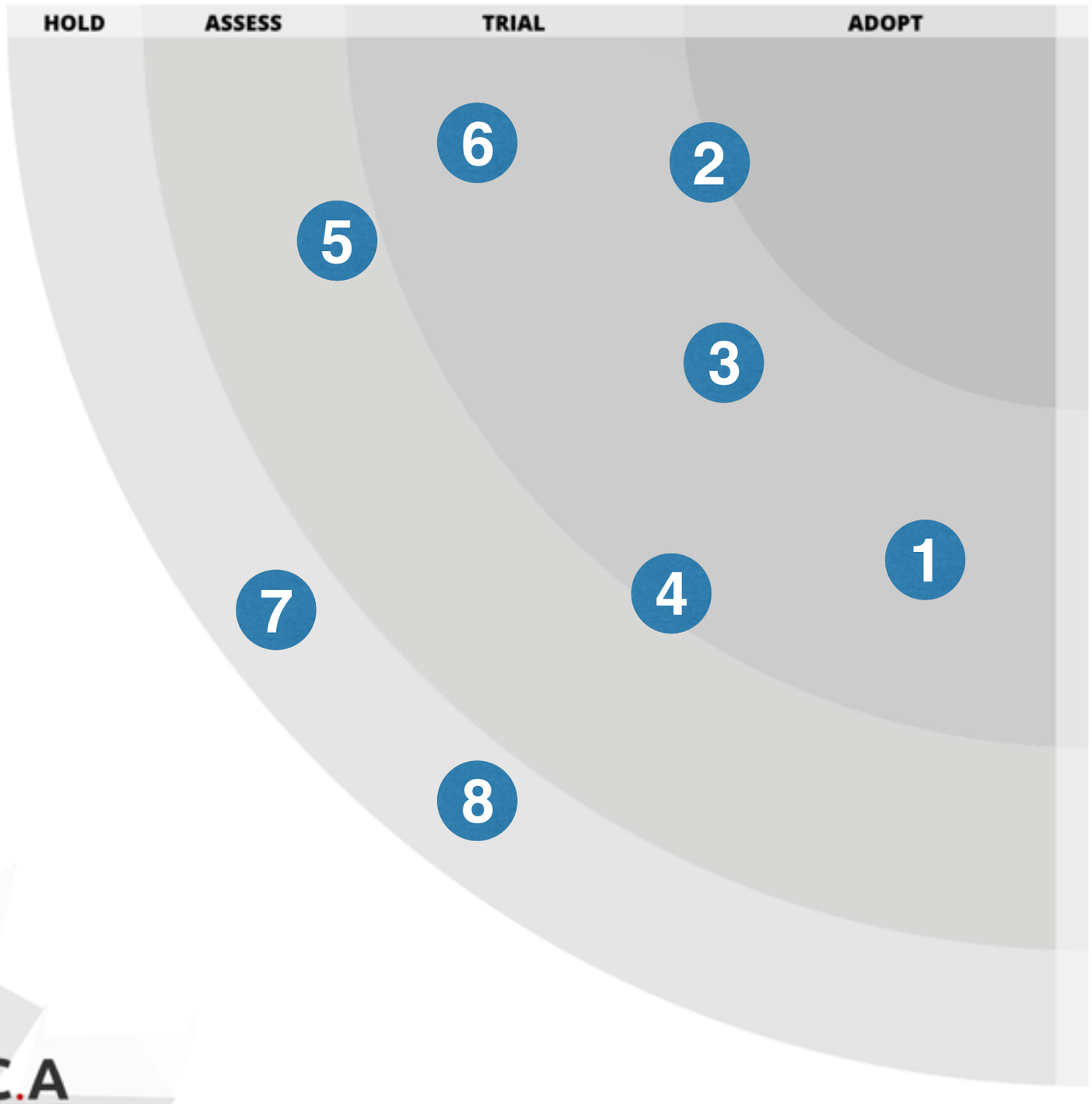
A.C.A MOBILE

Community efforts

MOCKTI TISHADOW TITANIUM-JASMINE TI-MOCHA TI02
PROXYQUIRE ...

A.C.A MOBILE

Stack Radar



Radar items

1. MockTi
2. TiShadow
3. Titanium-Jasmine
4. TiMocha
5. Tio2
6. ProxyQuire
7. Ti Alloy Jasmine Testing
8. TiJasmine

A.C.A MOBILE

mobile.aca-it.be/madstack

Didn’t do the **trick**

A.C.A MOBILE

Always run **in** the **container**

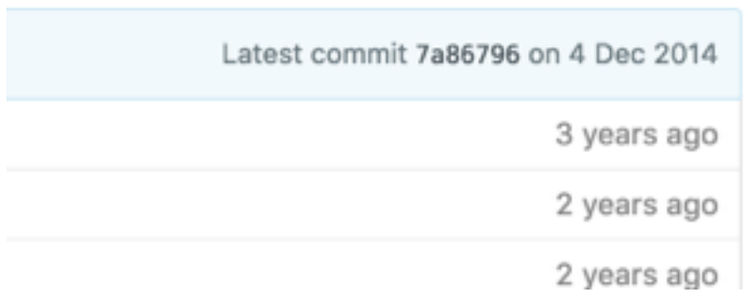
titanium-jasmine puts together Jasmine and Titanium Mobile, so you can write Jasmine tests that will run inside your iPhone, iPad and Android applications. **titanium-jasmine** is a **really** simple testing framework that will help you test your Titanium Mobile applications properly.

Instead of trying to mock all **Titanium.*** objects and functions, **titanium-jasmine** will run the tests on your simulator so you can keep using Titanium includes and helper functions on your code, thus making testing much easier.

A.C.A MOBILE

Simply **don’t work** (anymore)

or **abandoned**



A.C.A MOBILE

No mocking for **Ti** namespace

A.C.A MOBILE

No mocking for ***required*** dependencies

A.C.A MOBILE

Why so **important**?

A.C.A MOBILE

Testing is in our **DNA**

A.C.A MOBILE

Testing ensures **quality**

A.C.A MOBILE

Challenges

A.C.A MOBILE

Existing solutions are **slow** (due to the container)

A.C.A MOBILE

Existing solutions **lack isolation**

A.C.A MOBILE

With existing solutions, it is **impossible** to **fake device characteristics**

A.C.A MOBILE

Should be **maintainable** and **flexible** (support multiple Ti SDK versions)

A.C.A MOBILE

Full mocking of the **Ti** namespace

A.C.A MOBILE

Mocking of ***required*** resources

A.C.A MOBILE

Mocking of ***required*** resources

vabfs.network.client.js | jasmine.json | authenticate.js

```
1  var networkUtil = require('network.availability.util');
2
3  var DEFAULT_TIMEOUT = 10000;
```

A.C.A MOBILE

Context of ***this*** in **callback** should be **mockable**

A.C.A MOBILE

Context of ***this*** in **callback** should be **mockable**

```
if(!networkUtil.checkNetworkAvailability()){
    return;
}

Ti.API.debug('Creating network client');

var client = Ti.Network.createHTTPClient({
    onload : function(e) {
        var locationHeader = this.getResponseHeader('Location');
        if(!_.isUndefined(locationHeader) && _.isEqual(locationHeader.search("https://"), -1)) {
            locationHeader = locationHeader.replace("http://", "https://");
        }

        if (!_.isUndefined(successDataCallback)) {
```

A.C.A MOBILE

Our **SOLUTION**

A.C.A MOBILE

TiUnit

A.C.A MOBILE

Is **fast** and **executes outside** of the **container**

A.C.A MOBILE

Can **mock all dependencies**

A.C.A MOBILE

Required dependencies

(*MockRequire*)

A.C.A MOBILE

Generates a **mock** for **all** **functions**, **constants** and **properties** in the **Ti** namespace

A.C.A MOBILE

Has **strategies** for **testing**
Alloy.CFG and **Alloy.Globals**, **L**(‘language-macro’) and **$**,
callback testing, **mocking** the **this** context in **callback functions**

A.C.A MOBILE

TiUnit

Open Source

A.C.A MOBILE

Our point of **view** on **testing**

A.C.A MOBILE

Testing ensures **quality**

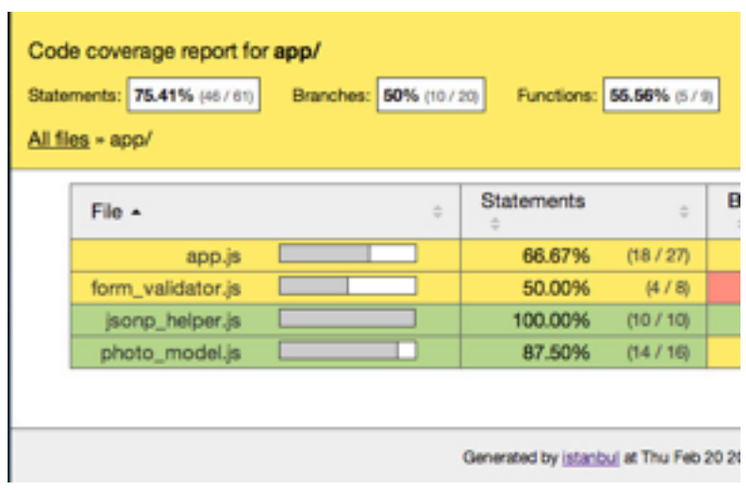
A.C.A MOBILE

Based on **3 pillars**

Jasmine + TiUnit + MockRequire

TiCalabash + Amazon DevFarm

Istanbul



Unit testing

Integration testing

Test coverage

A.C.A MOBILE

Reuse and **Automation**

A.C.A MOBILE

Mac OS X **Jenkins Slave**

Inject, Test, Coverage, Build, Sign and Deliver

A.C.A MOBILE

Before **getting started**

A.C.A MOBILE

TiUnit/MockRequire

- A **lightweight** pure javascript **environment**
- **Complete mock** of the **Ti** namespace
- **Mock *required*** libraries
- **Focuses** on **true isolated testing** of the unit-under-test
- **Best practices** to **unit test** your code

A.C.A MOBILE

Getting Started

```
Ti = require('tiunit/jsca.api.parser').parse();

MockRequire = require('tiunit/mockrequire');
```

A.C.A MOBILE

TiUnit - Injecting dependencies

controller

```
var UserManager = require('user.manager');
function _onAuthSuccess() { … }

module.exports.test: {
    _onAuthSuccess: _onAuthSuccess
}
```

controller-test

```
MockRequire = require('tiunit/mockrequire');

beforeEach(function () {
    MockRequire.addMock('user.manager', userManagerMock);
    controllerUnderTest = require('../app/controllers/authenticate');
});
```

A.C.A MOBILE

TiUnit - spy/expect on Ti namespace

```
spyOn(Ti.Network, ‘getNetworkType').and.returnValue(Ti.Network.NETWORK_NONE);
spyOn(Ti.App, ‘fireEvent’);
…
expect(Ti.App.fireEvent).toHaveBeenCalledWith("network:none");
```

A.C.A MOBILE

(!) Please use Ti instead of Titanium namespace in all your production code

TiUnit - $

test

```
$ = {
        btnAuthenticate: {
            addEventListener: function(){}
        },
        username: {
            addEventListener: function(){},
            getValue: function(){}
        }
    };

..
spyOn($.username, 'getValue').and.returnValue(USER_NAME);
..
expect(userManagerMock.setUserName).toHaveBeenCalledWith(USER_NAME);
```

We create a mock on $ for every view object we communicate with via our controller

We know this can require a lot of boilerplate, therefore we are working on a solution which will generate the controller mock for you.

A.C.A MOBILE

TiUnit - L (i18n macro)

function under test

```
function _onAuthLoginError() {
    Alloy.Globals.notifications.showError(L('my.message'));
}
```

test

```
L = function(s){
     return s;
};

it('should show error message when a login error has occured', function(){
    spyOn(Alloy.Globals.notifications, 'showError');
    controllerUnderTest.test._onAuthLoginError();

expect(Alloy.Globals.notifications.showError).toHaveBeenCalledWith('my.message');
});
```

A.C.A MOBILE

we don't spy on L directly (it's a macro and cannot by spied upon), instead we provide a mock implementation which returns our key. Now we can create a simple expectation on the surrounding function!

Writing testable callbacks (*this* context)

```
var client = Ti.Network.createHTTPClient({
      onload : function(e) {
          return this.getResponseHeader('Location');
      },
      onerror : function(e) {
          errorCallback && errorCallback(e, this.responseText);
      },
      timeout : (timeout || DEFAULT_TIMEOUT)
  });
```

A.C.A MOBILE

(!) Read the whole story on …

Writing testable callbacks (*this* context) - 2

```
function _prepareRequest(successCallback, successDataCallback, errorCallback, timeout) {
    if(!networkUtil.checkNetworkAvailability()){ return; }

    var that = this;
    var client = Ti.Network.createHTTPClient({
        onload : function(e){
            __onLoadCallback.apply(that, [e, successCallback, successDataCallback, this.getResponseHeader('Location'), this.responseDat
this.responseText]);
        },
        onerror : __onErrorCallback.apply(that, [e, errorCallback, this.responseText]),
        timeout : (timeout || DEFAULT_TIMEOUT)
    });
    return client;
}

function __onLoadCallback(e, successCallback, successDataCallback, locationHeader, responseData, responseText){ ...}

function __onErrorCallback(e, errorCallback, responseText){ errorCallback && errorCallback(e, responseText); }

function _post(successCallback, errorCallback, url, properties, timeout){
    var client = _prepareRequest(successCallback, undefined, errorCallback, timeout);
    ...
}

module.exports = {
    post: _post,
    test: {
        onloadCallback: __onLoadCallback,
        onErrorCallback: __onErrorCallback
    }
};
```

A.C.A MOBILE

Key takeaways

1. **Testing** ensures **quality**
2. **Light** and **Fast** environment
3. **Best practices** and **guidelines**
4. **It’s just the beginning**

Key links

1. **Project:** https://github.com/aca-mobile/ti-unit
2. **Documentation:** https://github.com/aca-mobile/ti-unit/wiki

What's **next**?

1. **Auto-generated** controller **mocks**
2. Maven build **hook** which **removes** those **.test.foo exports**
3. **Blogpost**

Lightweight **Unit Testing** for **Titanium**

Questions?