

# Assert Control Over Your Legacy Applications with TestBox

**By Ed Bartram**

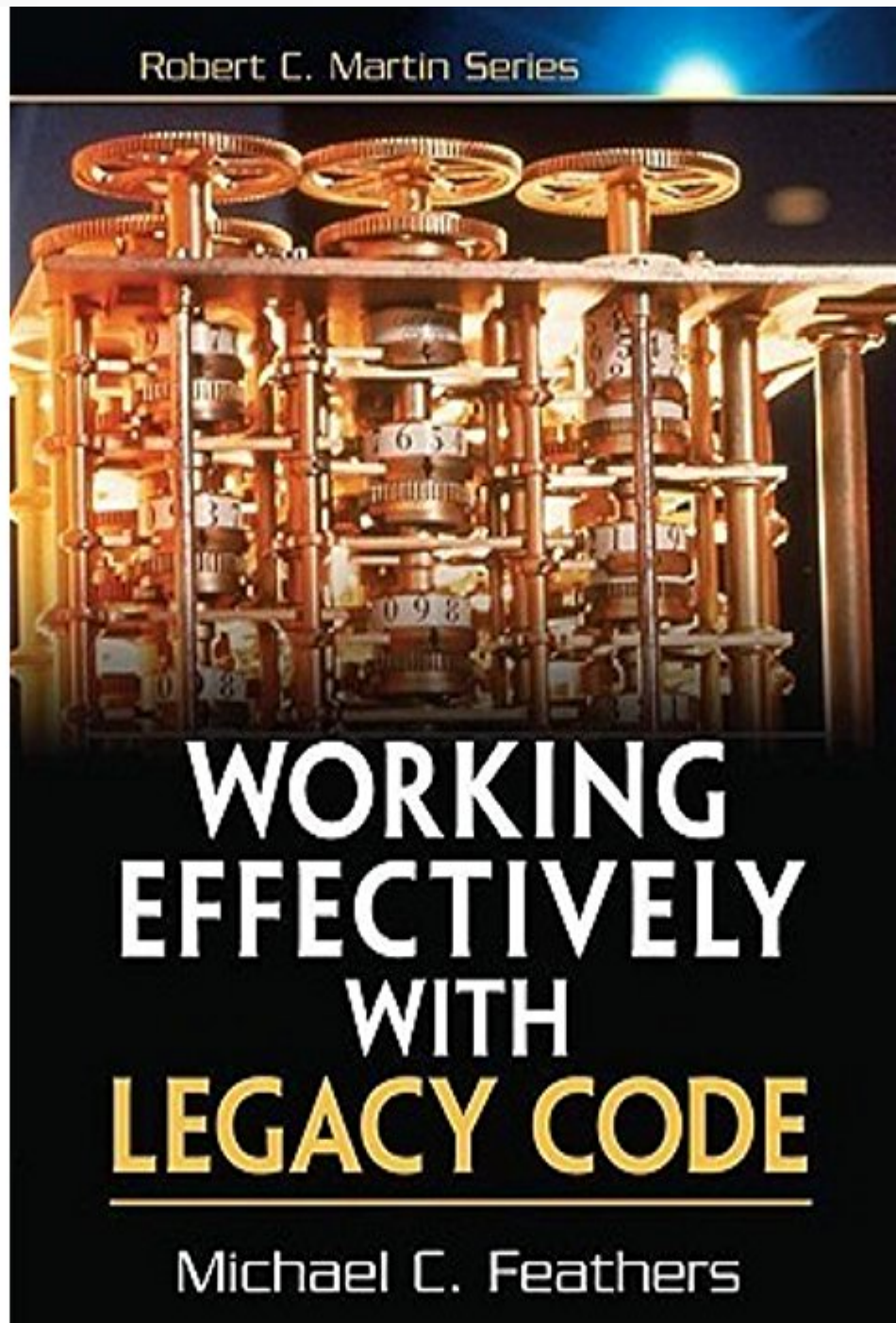
# Ed Bartram

- ColdFusion Developer since 2000 using version 4.5
- Previously co-manager of Chicagoland CFUG (CCFUG) and Nebraska CFUG (NECFUG)
- First Time Conference Speaker



# Assert Control Over Your Legacy Applications with TestBox

**By Ed Bartram**



*"To me, legacy code is simply code without tests."*

— Michael C. Feathers, Working Effectively with Legacy Code

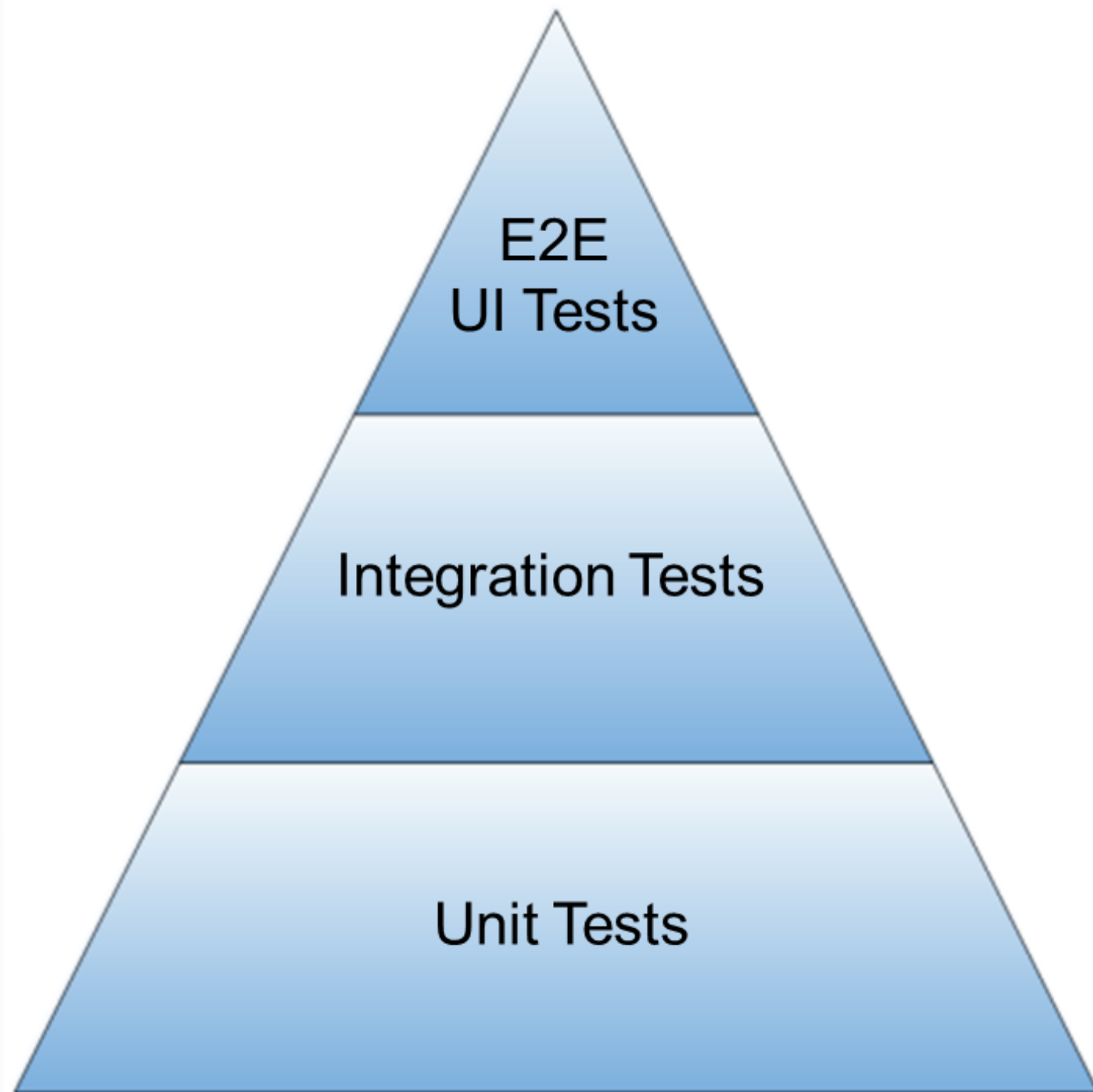
*“Regardless of what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.”*

— Norm Kerth, Project Retrospectives: A Handbook for Team Review

# What are tests? Why do we need them?







# Test Types Comparison

Type of Test	Complexity	Speed	Amount	Focus	Scope
Unit	Simple	Fast	Many	Implementation	Unit
Integration	Simple – Hard	Slow	Some	Implementation	Unit(s)
End to End (E2E) – Acceptance	Complex	Slow	Few	Behavior	Feature, Application
End to End (E2E) – Functional	Complex	Slow	Few	Behavior	Feature, Application



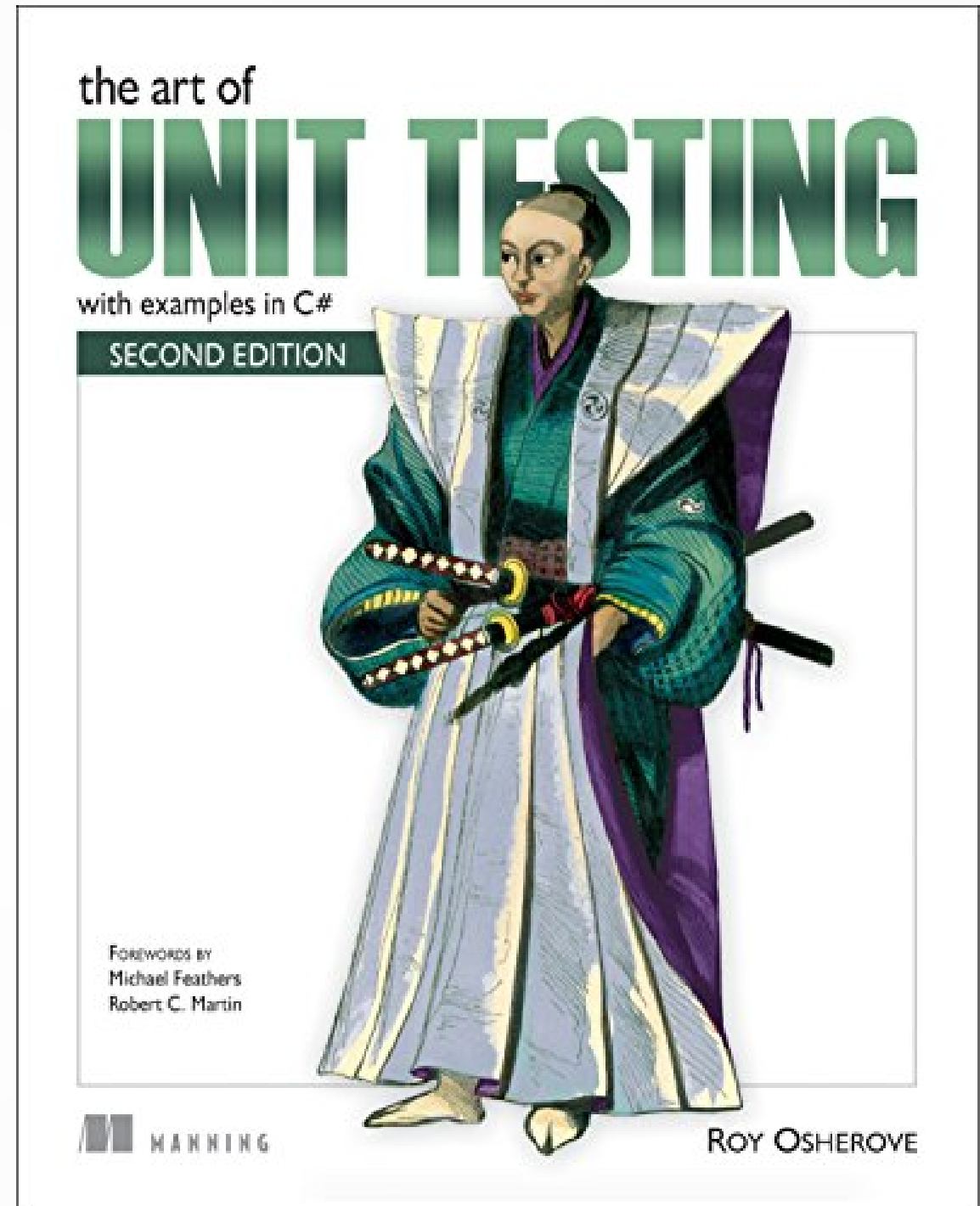
# Pre-Launch

- Separate Development Environment
- Version Control
- CFCs
- ColdFusion Framework
- Documented Coding Standards
- Peer Code Reviews



# Properties of a Unit Test

- Repeatable
- Easy to implement
- Relevant tomorrow
- Push button executable
- Runs quickly
- Consistent in results
- Full control of unit under test
- Fully isolated
- Pinpoint cause of failures



# Test Driven Development (TDD)

”Red, Green, Refactor”



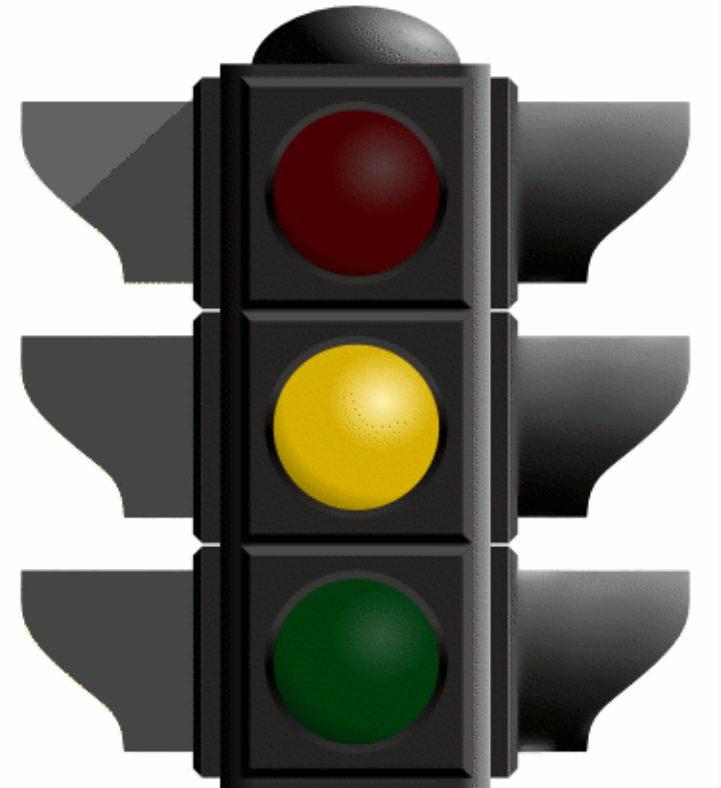
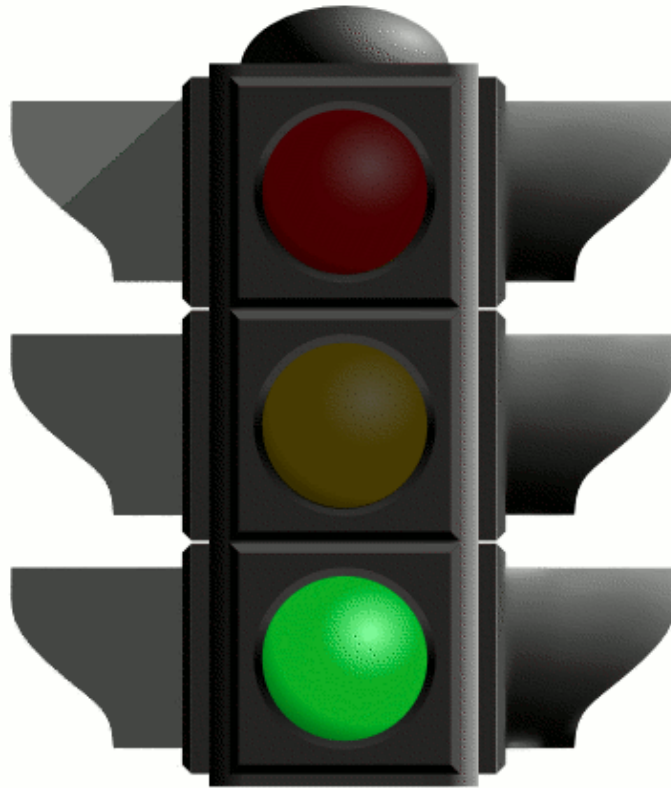
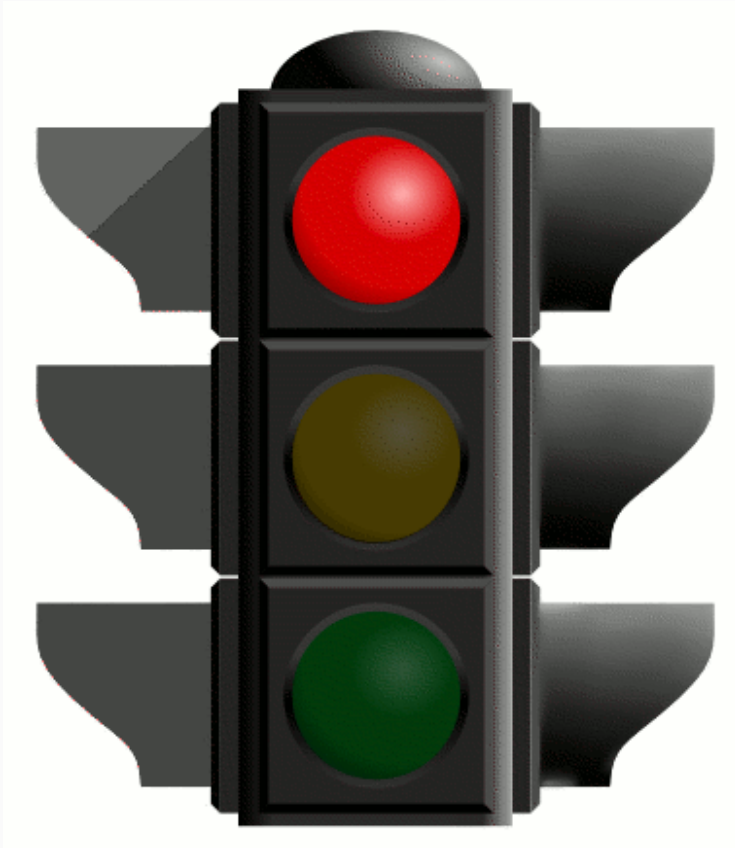
# But Wait!

**I already have code written without tests!**

**How do I follow TDD when making changes to my existing code?**



# TDD for Legacy Code



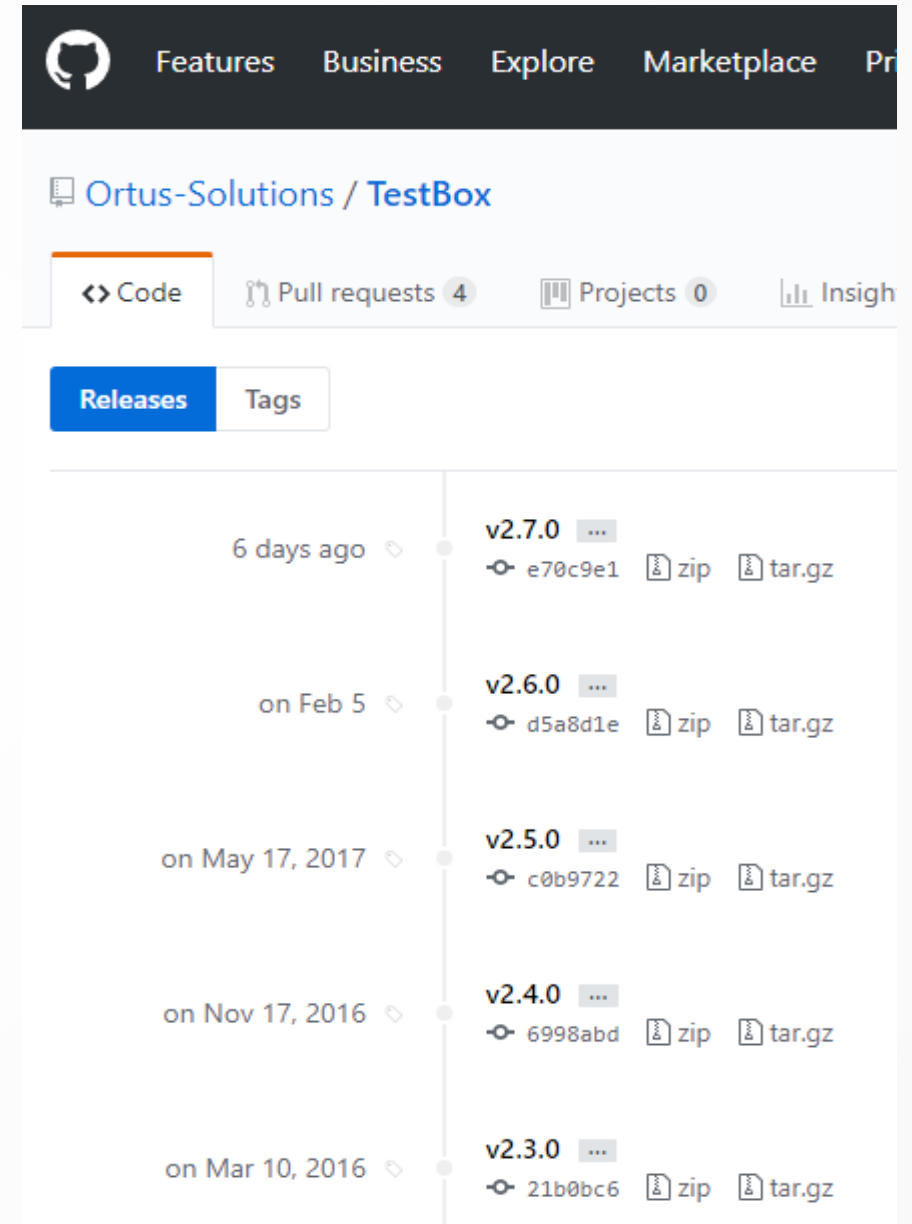
# Installing TestBox

```
1 // Install latest stable version
2 box install testbox
3
4 // Install bleeding edge version
5 box install testbox@be
6
```

- Install with CommandBox
- Download from ForgeBox and install manually unzipping file into /testbox folder
- Clone GitHub repository  
git clone git://github.com/ortus-solutions/testbox testbox
- Can install to another folder by creating a mapping:  
this.mappings[ "/testbox" ] = expandPath( "C:/frameworks/testbox/" )

# Supported CFML Engines

- ColdFusion 9, 10, or Railo 4.1 last supported on TestBox 2.3.0
- ColdFusion 11+ and Lucee 4.5+ currently supported
- Older versions are marked as releases on GitHub





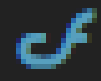
# Test Suites

- Tests go in /tests folder
- Recommend to split tests by type into unit, integration, and specs folders
- Mirror your site's structure in tests folders
- Mirror your objects using same name followed by the Test suffix
- Copy \testbox\test-browser\index.cfm to /tests and change rootMapping to "\tests\"

```
4 <!--- SETUP THE ROOTS OF THE BROWSER RIGHT HERE --->
5 <cfset rootMapping = "/testbox/tests/specs">
```

```
└─ model
  └─ store
    └─ ShoppingCart.cfc
    └─ ShoppingCartNew.cfc
  └─ testbox
  └─ tests
    └─ integration
      └─ model
        └─ store
          └─ ShoppingCartTest.cfc
      └─ specs
      └─ unit
        └─ model
          └─ store
            └─ ShoppingCartTest.cfc
            └─ ShoppingCartTestNew.cfc
        └─ index.cfm
```

# Extend the TestBox Framework

 shoppingCartTest.cfc ●

```
1  <cfcomponent extends="testbox.system.BaseSpec">
2  | ... |
3  </cfcomponent>
```

# Life Cycle Methods

```
3  <cffunction name="beforeTests" access="public" returntype="void">
4  </cffunction>
5
6  <cffunction name="afterTests" access="public" returntype="void">
7  </cffunction>
8
9  <cffunction name="setup" access="public" returntype="void">
10 |   <cfargument name="currentMethod" type="string" required="true">
11 </cffunction>
12
13 <cffunction name="tearDown" access="public" returntype="void">
14 |   <cfargument name="currentMethod" type="string" required="true">
15 </cffunction>
```

# setup() example

```
11 <cffunction name="setup" access="public" returntype="void">  
12     <cfargument name="currentMethod" type="string" required="true">  
13  
14     <cfset variables.shoppingCart = new model.store.ShoppingCart()>  
15 </cffunction>
```

# Unit Test Design Pattern

```
22 <cffunction name="testMethodName_WhenScenario" access="public" returntype="void">
23     <!--- Setup the Scenario --->
24     <cfset var scenario = "setup the scenario">
25
26     <!--- Exercise the Unit --->
27     <cfset var results = component.unit( scenario )>
28
29     <!--- Make Assertions based on the result --->
30     <cfset $assert.isTrue( results )>
31 </cffunction>
```

ShoppingCart.cfc ✕

```
1 <cfcomponent>
2
3     <cffunction name="addToCart" access="public" returnType="boolean">
4         <cfargument name="itemID" type="numeric" required="true">
5         <cfargument name="quantity" type="numeric" required="false" default=0>
6
7         <cfreturn arguments.quantity gt 0 ? true : false>
8     </cffunction>
```

ShoppingCart.cfc

ShoppingCartTest.cfc ✕

```
22 <cffunction name="testAddToCart_oneItem" access="public" returnType="void">
23     <cfset var scenario = {
24         itemID = 42,
25         quantity = 1
26     }>
27
28     <cfset var results = shoppingCart.addToCart( argumentCollection=scenario )>
29
30     <cfset $assert.isTrue( results )>
31 </cffunction>
```

ShoppingCart.cfc ✕

```
1 <cfcomponent>
2
3     <cffunction name="addToCart" access="public" returnType="boolean">
4         <cfargument name="itemID" type="numeric" required="true">
5         <cfargument name="quantity" type="numeric" required="false" default=0>
6
7         <cfreturn arguments.quantity gt 0 ? true : false>
8     </cffunction>
```

ShoppingCart.cfc

ShoppingCartTest.cfc ✕

```
33 <cffunction name="testAddToCart_noItem" access="public" returnType="void">
34     <cfset var scenario = {
35         itemID = 42,
36         quantity = 0
37     }>
38
39     <cfset var results = shoppingCart.addToCart( argumentCollection=scenario )>
40
41     <cfset $assert.isFalse( results )>
42 </cffunction>
```



# Assertions

\$assert.**isTrue**( actual )

\$assert.**isFalse**( actual )

\$assert.**isEqual**( expected, actual )

\$assert.**isNotEqual**( expected, actual )

\$assert.**null**( actual )

\$assert.**notNull**( actual )

\$assert.**typeOf**( “type”, actual )

\$assert.**notTypeOf**( “type”, actual )

\$assert.**instanceOf**( “type”, actual )

\$assert.**notInstanceOf**( “type”, actual )

\$assert.**isGT**( actual, target )

\$assert.**isGTE**( actual, target )

\$assert.**isLT**( actual, target )

\$assert.**isLTE**( actual, target )

\$assert.**isEmpty**( actual )

\$assert.**isNotEmpty**( actual )

\$assert.**lengthOf**( actual, length )

\$assert.**notLengthOf**( actual, length )

\$assert.**key**( actual, key )

\$assert.**notKey**( actual, key )

& many more

# Skips, Fails & Errors

ShoppingCart.cfc

ShoppingCartTest.cfc ✕

```
44 <cffunction name="testAddToCart_skip" access="public" returntype="void" skip>
45 </cffunction>
46
47 <cffunction name="testAddToCart_fail" access="public" returntype="void">
48 |   <cfset $assert.isEqual( "expected", "actual" )>
49 </cffunction>
50
51 <cffunction name="testAddToCart_error" access="public" returntype="void">
52 |   <cfset var results = shoppingCart.addToCart()>
53 </cffunction>
```

ShoppingCart.cfc

shoppingCartTest.cfc x

```
19
20     <cffunction name="testAddToCart_manyItems" access="public" returnType="void">
21         <cfset var scenario = setupScenario( itemID=42, quatntity=3 )>
22
23         <cfset var results = shoppingCart.addToCart( argumentCollection=scenario )>
24
25         <cfset $assert.isTrue( results )>
26     </cffunction>
27
28     <cffunction name="setupScenario" access="private" returnType="struct">
29         <cfargument name="itemID" type="numeric" required="true">
30         <cfargument name="quantity" type="numeric" required="false" default=0>
31
32         <cfreturn {
33             itemID = arguments.itemID,
34             quantity = arguments.quantity
35         }>
36     </cffunction>
```

# Test Runners

The screenshot shows a web browser window with the address bar displaying `127.0.0.1:52139/tests/unit/model/store/ShoppingCartTest.cfc?method=runRemote`. The page title is "Pass: 3 Fail: 1 Errors: 1". The main content area shows the TestBox v2.6.0+156 interface. It includes a "Filter Bundles..." search bar and a "Global Stats (22 ms)" section. The global stats show: [ Bundles/Suites/Specs: 1/1/6 ] [ Pass: 3 ] [ Failures: 1 ] [ Errors: 1 ] [ Skipped: 1 ] [ Reset ]. There are "Run All" and "Debug" buttons. Below this is the "tests.unit.model.store.ShoppingCartTest (20 ms)" section, which shows: [ Suites/Specs: 1/6 ] [ Pass: 3 ] [ Failures: 1 ] [ Errors: 1 ] [ Skipped: 1 ] [ Reset ]. The test results are listed as follows:

- +tests.unit.model.store.ShoppingCartTest (20 ms)
  - testAddToCart\_manyItems (1 ms)
  - testAddToCart\_error (10 ms) - The parameter itemID to function addToCart is required but was not passed in. +
  - testAddToCart\_noItem (0 ms)
  - testAddToCart\_fail (8 ms) - Expected [expected] but received [actual] +
  - testAddToCart\_oneItem (1 ms)

# Isolating Dependencies

ShoppingCart.cfc ✕

```
10  <cffunction name="saveCart" access="public" returnType="void">
11      <cfargument name="itemID" type="numeric" required="true">
12      <cfargument name="quantity" type="numeric" required="false" default=0>
13
14      <cfset var qReadCart = queryNew( "" )>
15
16      <cfquery name="qReadCart" datasource="dsn">
17          SELECT itemID
18          FROM shoppingCart
19          WHERE itemID = <cfqueryparam cfsqltype="integer" value="#arguments.itemID#">
20      </cfquery>
21
22      <cfif qReadCart.recordCount eq 0>
23          <cfset createCart( argumentCollection=arguments )>
24      <cfelse>
25          <cfset updateCart( argumentCollection=arguments )>
26      </cfif>
27  </cffunction>
```

# Refactoring

“... is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior”

— Martin Fowler, [refactoring.com](http://refactoring.com)

# Refactoring Example

```
14      <cfset var qReadCart = queryNew( "" )>
15
16      <cfquery name="qReadCart" datasource="dsn">
17          SELECT itemID
18          FROM shoppingCart
19          WHERE itemID = <cfqueryparam cfsqltype="integer" value="#arguments.itemID#">
20      </cfquery>
```

```
31      <cffunction name="readCart" access="private" returnType="query">
32          <cfargument name="itemID" type="numeric" required="true">
33
34          <cfset var qReadCart = queryNew( "" )>
35
36          <cfquery name="qReadCart">
37              SELECT itemID
38              FROM shoppingCart
39              WHERE itemID = <cfqueryparam cfsqltype="integer" value="#arguments.itemID#">
40          </cfquery>
41
42          <cfreturn qReadCart>
43      </cffunction>
```



# Refactoring Example Continued

ShoppingCartNew.cfc ✕

```
14      <cfset var results = {
15          |      action = "",
16          |      readCart = readCart( itemID=arguments.itemID )
17      |  }>
18      <cfset results.itemID = results.readCart.itemID>
19
20      <cfif val( results.readCart.itemID ) eq 0>
21          |      <cfset results.itemID = createCart( argumentCollection=arguments )>
22          |      <cfset results.action = "create">
23      <cfelse>
24          |      <cfset updateCart( argumentCollection=arguments )>
25          |      <cfset results.action = "update">
26      </cfif>
27
28      <cfreturn results>
29  </cffunction>
```

# Test Doubles

**Test Stub**

**Mock Object**

**Test Spy**

**Fake Object**

**Dummy Object**

# Test Double Example

ShoppingCartNew.cfc

ShoppingCartTestNew.cfc ✕

```
23  <cffunction name="testSaveCart_insert" access="public" returntype="void">
24      <cfset var scenario = {
25          |   itemID = 99,
26          |   quantity = 1
27      }>
28
29      <cfset var readCart = querySim( "itemID
30      " )>
31
32      <cfset shoppingCart.$( method="readCart", returns=readCart )>
33      <cfset shoppingCart.$( method="createCart", returns=scenario.itemID )>
34
35      <cfset var results = shoppingCart.saveCart( argumentCollection=scenario )>
36
37      <cfset $assert.key( results, "action" )>
38      <cfset $assert.isEqual( "create", results.action )>
39      <cfset $assert.key( results, "itemID" )>
40      <cfset $assert.isEqual( scenario.itemID, results.itemID )>
41  </cffunction>
```

## **Properties of Unit Tests**

- Repeatable
- Easy to implement
- Relevant tomorrow
- Push button executable
- Runs quickly
- Consistent in results
- Full control of unit under test
- Fully isolated
- Pinpoint cause of failures

## **Unit Test Definition**

*“A unit test is an automated piece of code that invokes the unit of work being tested, and then checks some assumptions about a single end result of that unit. A unit test is almost always written using a unit testing framework. It can be written easily and runs quickly. It's trustworthy, readable, and maintainable. It's consistent in its results as long as production code hasn't changed.”*

*“Integration testing [as] testing a unit of work without having full control over all of it and using one or more of its real dependencies, such as time, network, database, threads, random number generators, and so on.”*

*— Roy Oshero, The Art of Unit Testing*

```
12 <cffunction name="testReadCart" access="public" returntype="void">
13     <cfset var error = false>
14     <cfset var quantity = 99>
15     <cfset var results = false>
16
17     <cftransaction action="begin">
18         <cftry>
19             <cfset var qInsertShoppingCart = queryNew( "" )>
20
21             <cfquery result="qInsertShoppingCart" datasource="dsn">
22                 INSERT INTO shoppingCart ( quantity )
23                 VALUES ( <cfqueryparam cfsqltype="integer" value="#arguments.quantity#"> )
24             </cfquery>
25
26             <cfset makePublic( shoppingCart, "readCart" )>
27             <cfset results = shoppingCart.readCart( itemID=qInsertShoppingCart.GENERATEDKEY )>
28
29             <cfcatch type="any">
30                 <cfset debug( cfcatch )>
31                 <cfset error = true>
32             </cfcatch>
33
34             <cffinally>
35                 <cftransaction action="rollback">
36             </cffinally>
37         </cftry>
38     </cftransaction>
39
40     <cfset $assert.isFalse( error )>
41     <cfset $assert.isTrue( results )>
42 </cffunction>
```

IntGcThCeocoEdedChHcPrPa10ImxlujcMiTeUcc352WiWLoTeX

127.0.0.1:52139/tests/integration/model/store/ShoppingCartTest.cfc?method=runRemote

☆

T

TestBox v2.6.0+156

Filter Bundles...

Global stats (51 ms)

[ Bundles/Suites/Specs: 1/1/1 ] [ Pass: 0 ] [ Failures: 1 ] [ Errors: 0 ] [ Skipped: 0 ] [ Reset ]

Run All

Debug

tests.integration.model.store.ShoppingCartTest (47 ms)

[ Suites/Specs: 1/1 ] [ Pass: 0 ] [ Failures: 1 ] [ Errors: 0 ] [ Skipped: 0 ] [ Reset ]

+Shopping Cart Integration Tests (47 ms)

testReadCart (47 ms) - Expected [true] to be false +

tests.integration.model.store.shoppingcarttest\_cfc\$cf.udfCall(/tests/integration/model/store/ShoppingCartTest.cfc:44)

42: </cftransaction>

43:

44: <cfset \$assert.isFalse( error )>

45: <cfset \$assert.isTrue( results )>

46: </cffunction>

Debug Stream +

The following data was collected in order as your tests ran via the `debug()` method:

testReadCart

testReadCart - 4/12/18 at 12:12:43 PM CDT

Catch

Entries: 15



# Do's

- Never make a change to code until you have a working unit test.
- Test all logic branches including conditionals, loops, calculations, or any other decision making code.
- In an MVC framework, you only write tests for your Model, this is where all your logic and external dependencies should go. Your handler should be brain dead and contain zero logic

# Don'ts

- Chaining methods AKA “Train Wrecks”  
`getThis().getThat().getAnotherThing()`  
they violate the Law of Demeter
- Changing persistent data
- Running slow
- Tests depending or affecting other tests

# Homework

- querySim()
- runners
- reporting
- annotations
- custom assertions
- mockbox
- IDE Integration  
CF Builder, Sublime, and VS Code
- Test Doubles
- Asynchronous Tests



- Behavior Driven Development (BDD)
- Automated Builds
- Continuous Integration
- Code Metrics

# Summary

- Refactor your Legacy Code using Tests
- Focus on Unit Testing and add Integration and E2E tests as needed
- Write Unit Tests using properties and definition
- TDD – Always write your tests before writing new code or touching existing code
- Use a Testing Framework like TestBox
- Isolate your dependencies in your code and use Test Doubles in your tests
- Refactoring is restructuring code without changing its external behavior

# Thank You!

## Ed Bartram

Web: [edbartram.com](http://edbartram.com)

Slack: [edbartram](#)

GitHub: [edbartram](#)

Twitter: [@edbartram](#)

LinkedIn: [in/edbartram](#)