Article
Henry Pereira · Apr 11, 2019 10m read

Open Exchange

Brief introduction to Test Driven Development with Caché and CosFaker

Hello everyone,

I was first introduced to TDD almost 9 year ago, and I immediately fell in love with it.
Nowadays it's become very popular but, unfortunately, I see that many companies don't use it. Moreover, many developers don't even know what it is exactly or how to use it, mainly beginners.

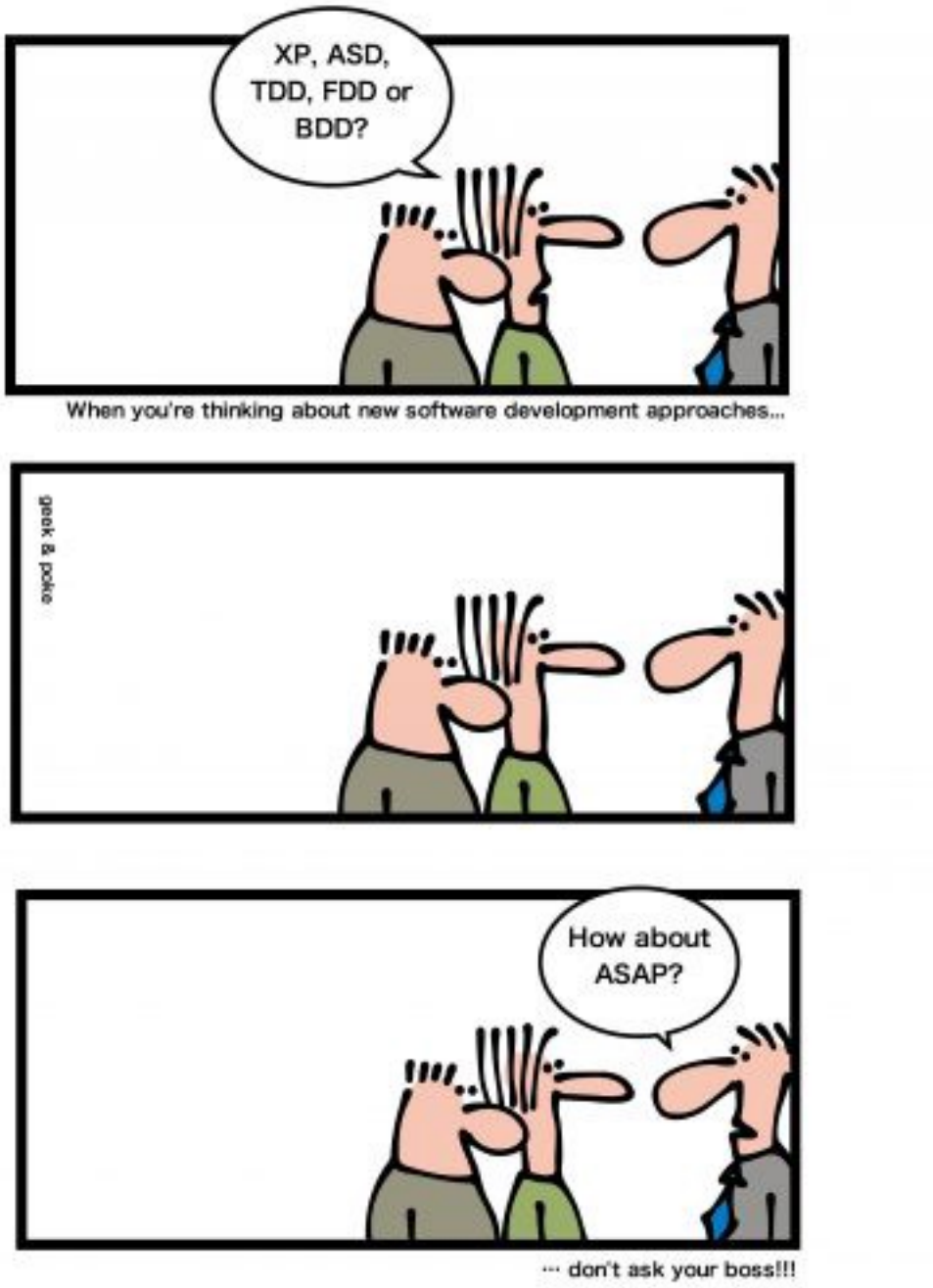
Overview

My goal with this article is to show how to use TDD with %UnitTest. I will show my workflow and explain how to use cosFaker, one of my first projects, which I created using Caché and recently uploaded to OpenExchange.

So buckle up and let's go.

What is TDD?

Test Driven Development (TDD) can be defined as a programming practice that instructs developers to write new code only if an automated test has failed.
There are tons of articles, lectures, talks, whatever, about its advantages and all of them are correct.
Your code is born already tested, you ensure that your system actually meets the requirements defined for it avoiding over-engineering, and you have constant feedback.



So why not use TDD? What is the problem with TDD? The answer is simple: the Cost! It costs a lot!
Because you have to write more lines of code with TDD, it is a slow process. But with TDD you have a final cost of creating a product NOW, without having to add to it in the future.
If you run the tests all the time you find errors early, thus reducing the cost of their correction.
So my advice: Just Do it!

Setup

InterSystems has a documentation and tutorial on how to use %UnitTest, you can read it here.

I use vscode to develop. This way I create a separate folder for tests. I add my project code path to UnitTestRoot and when I execute tests I pass the name of the test subfolder. And I always pass the qualifier loadudl

```
Set ^UnitTestRoot = "~/code"

  Do ##class(%UnitTest.Manager).RunTest("myPack","/loadudl")
```

Steps

Probably you have heard about the famous TDD cycle: red ➡ green ➡ refactor. You write a test that fails, you write a simple production code to make it pass and you refactor the production code.
So let's get the hands dirty and create a class to do some maths and another class to test it. The latter class shall extend %UnitTest.TestCase.
Now let's create a ClassMethod to return a square of an integer number:

```
Class Production.Math

{


ClassMethod Square(pValue As %Integer) As %Integer

{

}


}
```

And test what will happen if we pass 2. It must return 4.

```
Class TDD.Math Extends %UnitTest.TestCase

{

Method TestSquare()

{

    Do $$$AssertEquals(##class(Production.Math).Square(2), 4)

}

}
```

If you run:

```
Do ##class(%UnitTest.Manager).RunTest("TDD","/loadudl")
```

the test will Fail

Red! The next step is to make it Green.
To make it work let ´s return 4 as a result of execution of our Square method.

```
Class Production.Math

{

ClassMethod Square(pValue As %Integer) As %Integer
```

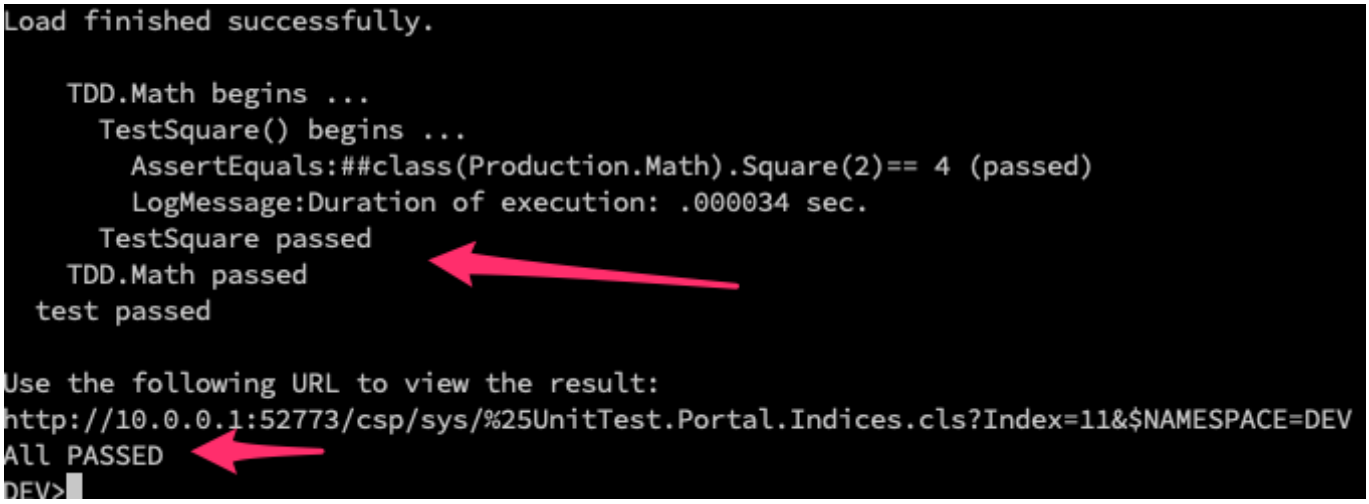
```
{

  Quit 4

}


}
```

and rerun our test.



Probably you aren't very happy with this solution, because it actually works for only one scenario. Fine! Let's take the next step. Let's create another test scenario, now sending a negative number.

```
Class TDD.Math Extends %UnitTest.TestCase

{


Method TestSquare()

{

    Do $$$AssertEquals(##class(Production.Math).Square(2), 4)

}


Method TestSquareNegativeNumber()

{

    Do $$$AssertEquals(##class(Production.Math).Square(-3), 9)

}
```

```
}
```

When we run the test:

```
Load finished successfully.

    TDD.Math begins ...
      TestSquare() begins ...
        AssertEquals:##class(Production.Math).Square(2)== 4 (passed)
        LogMessage:Duration of execution: .000024 sec.
      TestSquare passed
      TestSquareNegativeNumber() begins ...
AssertEquals:##class(Production.Math).Square(-3)== 9 was '4' (failed)  <<==== **FAILED**
test:TDD.Math:TestSquareNegativeNumber:
        LogMessage:Duration of execution: .000019 sec.
      TestSquareNegativeNumber failed
    TDD.Math failed
  test failed
```

it will Fail again, so let's refactor the production code:

```
Class Production.Math

{

ClassMethod Square(pValue As %Integer) As %Integer

{

  Quit pValue * pValue

}

}
```

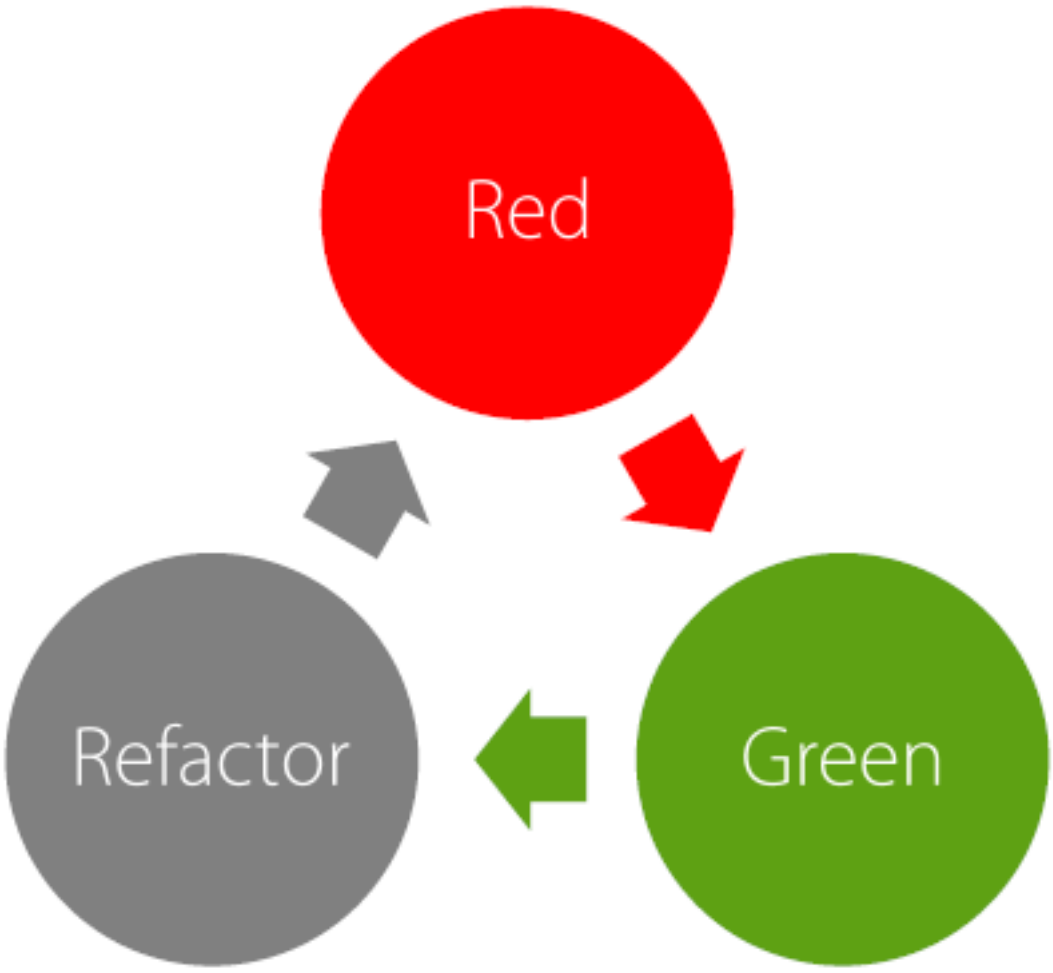
and rerun our tests:

```
Load finished successfully.

    TDD.Math begins ...
      TestSquare() begins ...
        AssertEquals:##class(Production.Math).Square(2)== 4 (passed)
        LogMessage:Duration of execution: .000033 sec.
      TestSquare passed
      TestSquareNegativeNumber() begins ...
        AssertEquals:##class(Production.Math).Square(-3)== 9 (passed)
        LogMessage:Duration of execution: .000016 sec.
      TestSquareNegativeNumber passed
    TDD.Math passed
  test passed

Use the following URL to view the result:
http://10.0.0.1:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=13&$NAMESPACE=D
All PASSED
```

Now everything works fine... This is the cycle of TDD, in small steps.



You must be asking yourself, why do I have to follow these steps? Why do I have to see the test fail?
I've worked in teams that wrote the production code and only later wrote the tests. But I prefer to follow these baby

steps for the following reasons:
Uncle Bob (Robert C. Martin) said that writing tests after writing the code is not TDD and instead is called "a waste of time".
Another detail, when I see the test fail, and later I see it pass, I'm testing the test.
Your test is still a code; and it may contain mistakes too. And the way to test it is to guarantee that it fails and passes when it needs to fail and pass. This way you've "tested the test".

cosFaker

To write good tests you might need to generate test data first. One way to do it is to generate a dump of data and use it in your tests.
Another way is to use cosFaker to easily generate fake data when you need it. https://openexchange.intersystems.com/package/CosFaker

You just need to download the xml file, after that go to Management Portal -> System Explorer -> Classes -> Import. Select the xml file to import, or drag the file in Studio.
Also you can import it using Terminal

```
Do $system.OBJ.Load("yourpath/cosFaker.vX.X.X.xml","ck")
```

Localization

cosFaker will add locales files in default CSP application folder. For now there are only two languages: English and Brazilian Portuguese (my native language).
The language of data is chosen according to configuration of your Caché.
The localization of cosFaker is an ongoing process, if you want to help, please don't hesitate to create a localized provider for your own locale and submit a Pull Request.
With cosFaker you can generate random Words, Paragraphs, Phone numbers, Names, Addresses, Emails, Prices, Product Names, Dates, Hexadecimal color codes... etc.

All methods are grouped by subject in classes, i.e. to generate a Latitude you call the method Latitude in class Address

```
 Write ##class(cosFaker.Address).Latitude()

-37.6806
```

You can also generate Json for your tests

```
Write ##class(cosFaker.JSON).GetDataJSONFromJSON("{ip:'ipv4',created_at:'date.backwar
d 40',login:'username', text: 'words 3'}")

{
    "created_at":"2019-03-08",
    "ip":"95.226.124.187",
    "login":"john46",
```

```
    "text":"temporibus fugit deserunt"
}
```

Here is a full list of **cosFaker** classes and methods:

- **cosFaker.Address**
  - StreetSuffix
  - StreetPrefix
  - PostCode
  - StreetName
  - Latitude
    - *Output:* -54.7274
  - Longitude
    - *Output:* -43.9504
  - Capital(Location = “”)
  - State(FullName = 0)
  - City(State = “”)
  - Country(Abrev = 0)
  - SecondaryAddress
  - BuildingNumber
- **cosFaker.App**
  - FunctionName(Group= “”, Separator = “”)
  - AppAction(Group= “”)
  - AppType
- **cosFaker.Coffee**
  - BlendName
    - *Output:* Cascara Cake
  - Variety
    - *Output:* Mundo Novo
  - Notes
    - *Output:* crisp, slick, nutella, potato defect!, red apple
  - Origin
    - *Output:* Rulindo, Rwanda
- **cosFaker.Color**
  - Hexadecimal
    - *Output:* #A50BD7
  - RGB
    - *Output:* 189,180,195
  - Name
- **cosFaker.Commerce**
  - ProductName
  - Product
  - PromotionCode
  - Color
  - Department
  - Price(Min = 0, Max = 1000, Dec = 2, Symbol = “”)
    - *Output:* 556.88
  - CNPJ(Pretty = 1)
    - CNPJ is the Brazilian National Registry of Legal Entities
    - *Output:* 44.383.315/0001-30
- **cosFaker.Company**
  - Name
  - Profession
  - Industry
- **cosFaker.Dates**
  - Forward(Days = 365, Format = 3)

    - Backward(Days = 365, Format = 3)
- **cosFaker.DragonBall**
    - Character
        - *Output:* Gogeta
- **cosFaker.File**
    - Extension
        - *Output:* txt
    - MimeType
        - *Output:* application/font-woff
    - Filename(Dir = “”, Name = “”, Ext = “”, DirectorySeparator = “/”)
        - *Output:* repellat.architecto.aut/aliquid.gif
- **cosFaker.Finance**
    - Amount(Min = 0, Max = 10000, Dec = 2, Separator= “,”, Symbol = “”)
        - *Output:* 3949,18
    - CreditCard(Type = “”)
        - *Output:* 3476-581511-6349
    - BitcoinAddress(Min = 24, Max = 34)
        - *Output:* 1WoR6fYvsE8gNXkBkeXvNqGECPUZ
- **cosFaker.Game**
    - MortalKombat
        - *Output:* Raiden
    - StreetFighter
        - *Output:* Akuma
    - Card(Abrev = 0)
        - *Output:* 5 of Diamonds
- **cosFaker.Internet**
    - UserName(FirstName = “”, LastName = “”)
    - Email(FirstName = “”, LastName = “”, Provider = “”)
    - Protocol
        - *Output:* http
    - DomainWord
    - DomainName
    - Url
    - Avatar(Size = “”)
        - *Output:* http://www.avatarpro.biz/avatar?s=150
    - Slug(Words = “”, Glue = “”)
    - IPV4
        - *Output:* 226.7.213.228
    - IPV6
        - *Output:* 0532:0b70:35f6:00fd:041f:5655:74c8:83fe
    - MAC
        - *Output:* 73:B0:82:D0:BC:70
- **cosFaker.JSON**
    - GetDataOBJFromJSON(Json = “” // JSON template string to create data)
        - *Parameter Example*: "{dates:'5 date'}"
        - *Output*: {"dates":["2019-02-19","2019-12-21","2018-07-02","2017-05-25","2016-08-14"]}
- **cosFaker.Job**
    - Title
    - Field
    - Skills
- **cosFaker.Lorem**
    - Word
    - Words(Num = “”)
    - Sentence(WordCount = “”, Min = 3, Max = 10)
        - *Output:* Sapiente et accusamus reiciendis iure qui est.
    - Sentences(SentenceCount = “”, Separator = “”)
    - Paragraph(SentenceCount = “”)
    - Paragraphs(ParagraphCount = “”, Separator = “”)
    - Lines(LineCount = “”)

    - Text(Times = 1)
    - Hipster(ParagraphCount = “”, Separator = “”)
- **cosFaker.Name**
    - FirstName(Gender = “”)
    - LastName
    - FullName(Gender = “”)
    - Suffix
- **cosFaker.Person**
    - cpf(Pretty = 1)
        - CPF is the Brazilian Social Security Number
        - *Output:* 469.655.208-09
- **cosFaker.Phone**
    - PhoneNumber(Area = 1)
        - *Output:* (36) 9560-9757
    - CellPhone(Area = 1)
        - *Output:* (77) 94497-9538
    - AreaCode
        - *Output:* 17
- **cosFaker.Pokemon**
    - Pokemon(EvolvesFrom = “”)
        - *Output:* Kingdra
- **cosFaker.StarWars**
    - Characters
        - *Output:* Darth Vader
    - Droids
        - *Output:* C-3PO
    - Planets
        - *Output:* Takodana
    - Quotes
        - *Output:* Only at the end do you realize the power of the Dark Side.
    - Species
        - *Output:* Hutt
    - Vehicles
        - *Output:* ATT Battle Tank
    - WookieWords
        - *Output:* nng
    - WookieSentence(SentenceCount = “”)
        - *Output:* ruh ga ru hnn-rowr mumwa ru ru mumwa.
- **cosFaker.UFC**
    - Category
        - *Output:* Middleweight
    - Fighter(Category = “”, Country = “”, WithISOCountry = 0)
        - *Output:* Dmitry Poberezhets
    - Featherweight(Country = “”)
        - *Output:* Yair Rodriguez
    - Middleweight(Country = “”)
        - *Output:* Elias Theodorou
    - Welterweight(Country = “”)
        - *Output:* Charlie Ward
    - Lightweight(Country = “”)
        - *Output:* Tae Hyun Bang
    - Bantamweight(Country = “”)
        - *Output:* Alejandro Pérez
    - Flyweight(Country = “”)
        - *Output:* Ben Nguyen
    - Heavyweight(Country = “”)
        - *Output:* Francis Ngannou
    - LightHeavyweight(Country = “”)
        - *Output:* Paul Craig

- Nickname(Fighter = “”)
  - *Output:* Abacus

Let's create a class for user with a method that returns his username, that will be FirstName concatenated with LastName.

```
Class Production.User Extends %RegisteredObject

{

Property FirstName As %String;

Property LastName As %String;

Method Username() As %String

{

}

}
```

```
Class TDD.User Extends %UnitTest.TestCase

{

Method TestUsername()

{

  Set firstName = ##class(cosFaker.Name).FirstName(),

    lastName = ##class(cosFaker.Name).LastName(),

    user = ##class(Production.User).%New(),

    user.FirstName = firstName,

    user.LastName = lastName

  Do $$$AssertEquals(user.Username(), firstName _ "." _ lastName)

}

}
```

```
    TDD.Math passed
    TDD.User begins ...
      TestUsername() begins ...
LogStateStatus:0:TestUsername: ERROR #5002: ObjectScript error: <COMMAND>zTestUsername+6^TDD.User.1 *Functi
on must return a value at zUsername+1^Production.User.1  <<==== **FAILED**
test:TDD.User:TestUsername:
      TestUsername failed
    TDD.User failed
  test failed

Use the following URL to view the result:
http://10.0.0.1:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=19&$NAMESPACE=DEV
```

Refactoring:

```
Class Production.User Extends %RegisteredObject

{

Property FirstName As %String;

Property LastName As %String;

Method Username() As %String

{

  Quit ..FirstName _ "." _ ..LastName

}

}
```

```
    TDD.Math passed
    TDD.User begins ...
      TestUsername() begins ...
        AssertEquals:user.Username()== firstName _ "." _ lastName (passed)
        LogMessage:Duration of execution: .001561 sec.
      TestUsername passed
    TDD.User passed
  test passed

Use the following URL to view the result:
http://10.0.0.1:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=20&$NAMESPACE=
All PASSED
```

Now we are going to add an account expiry date and validate it.

```
Class Production.User Extends %RegisteredObject

{
```

```
Property FirstName As %String;

Property LastName As %String;

Property AccountExpires As %Date;

Method Username() As %String
{
  Quit ..FirstName _ "." _ ..LastName
}

Method Expired() As %Boolean
{
}

}



Class TDD.User Extends %UnitTest.TestCase
{

Method TestUsername()
{
  Set firstName = ##class(cosFaker.Name).FirstName(),
    lastName = ##class(cosFaker.Name).LastName(),
    user = ##class(Production.User).%New(),
    user.FirstName = firstName,
    user.LastName = lastName
    Do $$$AssertEquals(user.Username(), firstName _ "." _ lastName)
}

Method TestWhenIsNotExpired() As %Status
{
```

```
  Set user = ##class(Production.User).%New(),

    user.AccountExpires = ##class(cosFaker.Dates).Forward(40)

  Do $$$AssertNotTrue(user.Expired())

}


}
```

```
    TDD.User begins ...
      TestUsername() begins ...
        AssertEquals:user.Username()== firstName _ "." _ lastName (passed)
        LogMessage:Duration of execution: .004456 sec.
      TestUsername passed
      TestWhenIsNotExpired() begins ...
LogStateStatus:0:TestWhenIsNotExpired: ERROR #5002: ObjectScript error: <COMMAND>zTestWhenIsNotExpired+3^TD
D.User.1 *Function must return a value at zExpired+1^Production.User.1  <<==== **FAILED**
test:TDD.User:TestWhenIsNotExpired:
      TestWhenIsNotExpired failed
    TDD.User failed
  test failed
```

Refactoring:

```
Method Expired() As %Boolean

{

  Quit ($system.SQL.DATEDIFF("dd", ..AccountExpires, +$Horolog) > 0)

}
```

```
    TDD.User begins ...
      TestUsername() begins ...
        AssertEquals:user.Username()== firstName _ "." _ lastName (passed)
        LogMessage:Duration of execution: .003934 sec.
      TestUsername passed
      TestWhenIsNotExpired() begins ...
        AssertNotTrue:user.Expired() (passed)
        LogMessage:Duration of execution: .000106 sec.
      TestWhenIsNotExpired passed
    TDD.User passed
  test passed
```

Now let's test when account is expired:

```
Method TestWhenIsExpired() As %Status

{

  Set user = ##class(Production.User).%New(),

    user.AccountExpires = ##class(cosFaker.Dates).Backward(40)
```

```
  Do $$$AssertTrue(user.Expired())

}
```

```
    TDD.User begins ...
      TestUsername() begins ...
        AssertEquals:user.Username()== firstName _ "." _ lastName (passed)
        LogMessage:Duration of execution: .00448 sec.
      TestUsername passed
      TestWhenIsExpired() begins ...
        AssertTrue:user.Expired() (passed)
        LogMessage:Duration of execution: .000105 sec.
      TestWhenIsExpired passed
      TestWhenIsNotExpired() begins ...
        AssertNotTrue:user.Expired() (passed)
        LogMessage:Duration of execution: .000072 sec.
      TestWhenIsNotExpired passed
    TDD.User passed
  test passed

Use the following URL to view the result:
http://10.0.0.1:52773/csp/sys/%25UnitTest.Portal.Indices.cls?Index=40&$NAMESPACE=DEV
All PASSED
DEV>
```

And Everything is green..

I know these are silly examples, but this way you will have simplicity not only in the code but also in the class design.

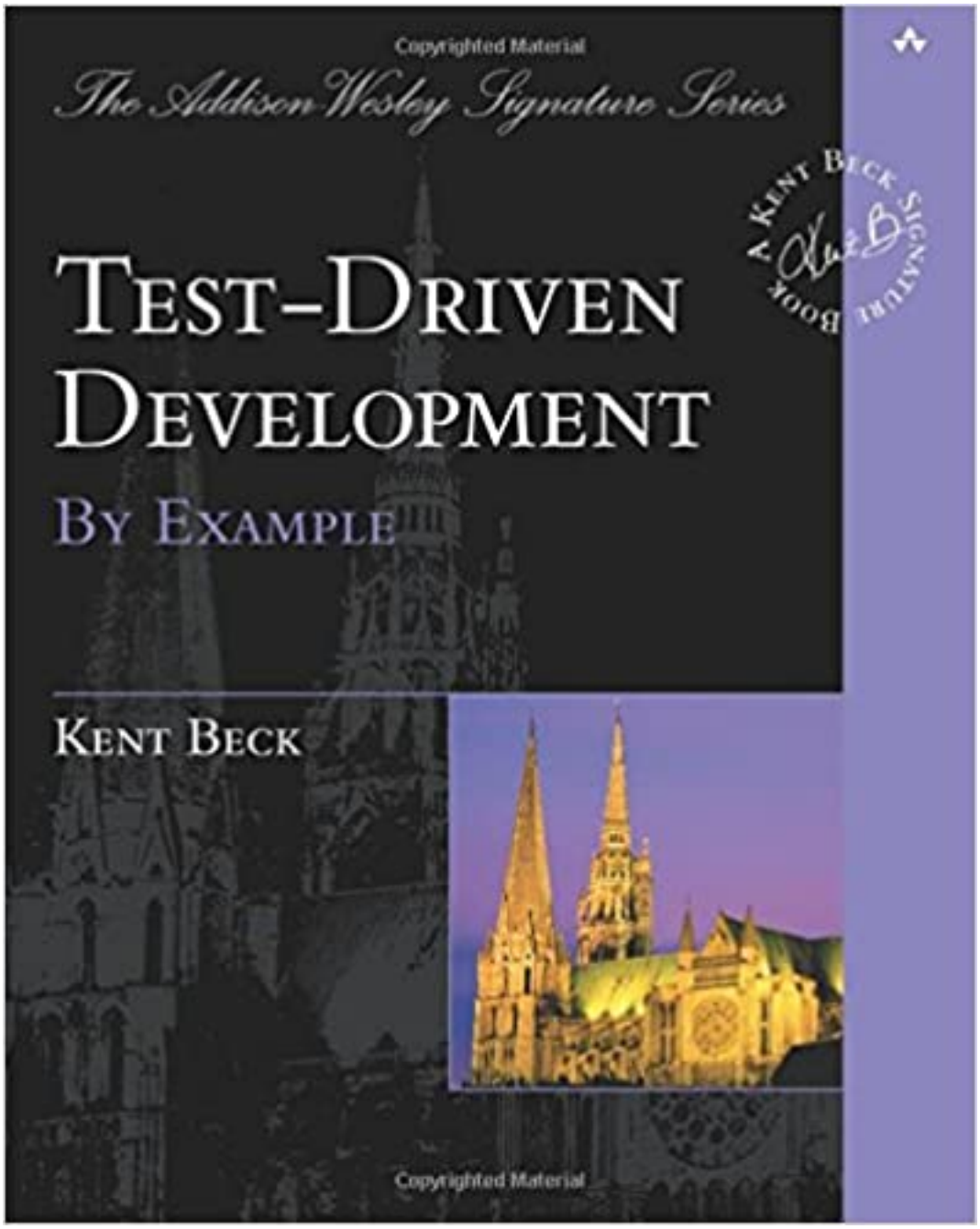
Conclusion

In this article you have learnt a little bit about Test Driven Development and how to use the %UnitTest class.
We also covered cosFaker and how to generate fake data for your tests.

There's much more to learn about tests and TDD, how to use these practices with legacy code, integration tests, acceptance tests (ATDD), bdd, etc...
If you want to know more about it, I strongly recommend 2 books:

Test Driven Development Teste e design no mundo real com Ruby - Mauricio Aniche, I really don't know if this book has english version. There are editions for Java, C#, Ruby and PHP. This book blew my mind with its awesomeness.

And of course, Kent Beck's book Test Driven Development by Example



Feel free to leave any comments or questions.
That's all folks

#Best Practices #Testing #Caché #InterSystems IRIS
Check the related application on InterSystems Open Exchange

Source URL: https://community.intersystems.com/post/brief-introduction-test-driven-development-cach%C3%A9-and-cosfaker