

A brief introduction to Test Driven Development

using Microsoft Excel and VBA

By Clarke Ching

www.clarkeching.com

1 Introduction

Test Driven Development (TDD) is one of the most extraordinary techniques to come out of eXtreme Programming (XP). It's an easy process to describe: write a failing test, make the test run, refactor the code so the design is cleaner, and then repeat for the next test. Essentially you are building your code-base in small incremental steps, building one test at a time. The beauty of TDD is that these small steps combined with Refactoring keep the code-base clean, well designed and malleable, so it is easy and cheap to work with. And even better, the tests act as a safety net -catching your mistakes quickly so that they are cheaper and easier to fix.

But describing the process isn't enough: it's only when you *do it* that you *truly get it*. But how do you *do* TDD, if you're not a programmer? How do you persuade your boss to let you do TDD, if he or she can't *truly get it*?

Here's a simple TDD example - converting integers into their roman number equivalent - that you or your boss can do easily using Microsoft Excel and Visual Basic for Applications (VBA). It's not intended to teach how to do TDD or how to program, but rather it's a hands on example where you can get a feel for the process and the possibilities.

All you need is Excel, an open mind and an hour or so to experiment. It will help if you've had a little programming experience ... but not much.

The process looks like this:

- You add test data into a spreadsheet (1=i, 2=ii, 3=iii, etc) one test at a time.
- You write a new Excel function using Visual Basic for Applications (VBA) to make each test work, as it is added .
- You set up conditional green and red formatting in the spreadsheet to tell you quickly and visually when the tests are passing and failing.

2 First test and testing environment

I fire up Excel and start with a blank spreadsheet.

- I type column headings in row 1 ("Integer" in Column A and "Roman" in column B).
- I add my first row of test data in row 2, using the test case that 1=i.

Sheet1:

	A	B
1	Integer	Roman
2	1	I

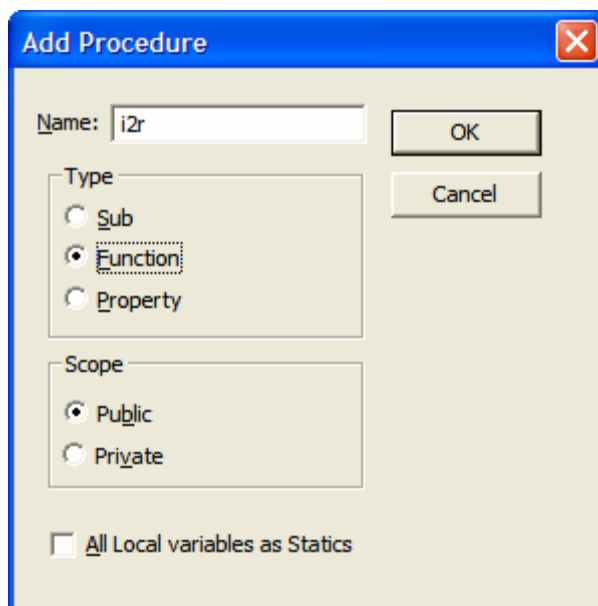
- Then I add another heading into cell C1 called “i2r” and I use in cell C2 I type the formula ‘=i2r(A2)’. This is the function that I’m going to build, test-case by test-case.

But, since the function i2r() doesn’t exist yet, Excel gives an error. This is the same as a compile error in, say, Java.

	A	B	C
1	Integer	Roman	i2r
2	I	I	#NAME?

I need to add a new function called i2r() before it will work.

- So I chose Tools/Macros/Visual Basic Editor which, unsurprisingly, opened up the Visual Basic editor. I could also have pressed ALT-F11. I then hunt around the menus and find INSERT/PROCEDURE which sounded handy - I presume that this is how I insert a new function (not just a procedure).
- Bad news: it’s greyed out.
- But, just underneath it is INSERT/MODULE which I bravely click and it gives me a new module. I check under the INSERT menu and PROCEDURE is now enabled. So I click on it and add in a new function (not procedure) - naming it i2r().



Then I add in an integer parameter to the function and (lazily) call it i. I also add in “Application.Volatile” to the top of the function which forces Excel to recalculate spreadsheet cells which use VBA code that has changed (it doesn’t do this automatically otherwise). You won’t believe how well this little trick is hidden in the Excel help files.

```
Public Function i2r(i As Integer)
Application.Volatile - don't forget to add this!
End Function
```

- I switch back to my spreadsheet and manually recalculate the spreadsheet by pressing F9. It shows me:

	A	B	C
1	Integer	Roman	i2r
2	1	I	0

My first test is now failing! This is progress. But, why is i2r returning 0? I look at the code and realise the VBA default must be to return 0.

- I switch to VBA, take a peek in the help files to learn how VBA returns parameters and then change the function so that it returns the string “I”.

```
Public Function i2r(i As Integer) As String
Application.Volatile
i2r = "I"
End Function
```

- Then I pop back to Excel. Do a manual “F9” recalc and hey-presto! my first test has passed.

	A	B	C
1	Integer	Roman	i2r
2	1	I	I

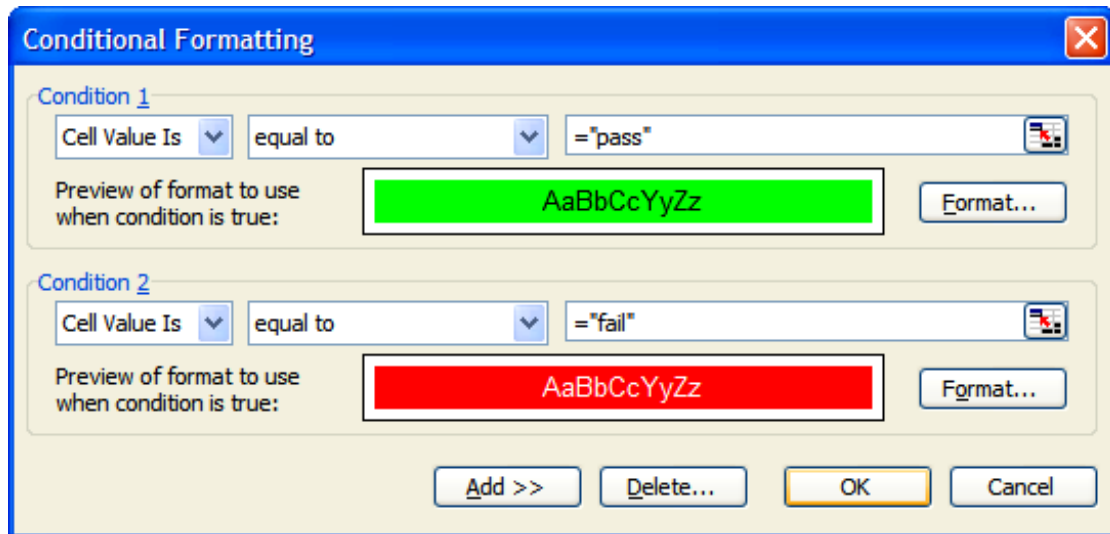
Phewww! Since I’m journaling my steps this has taken a while, but in normal circumstances I reckon it would take 1-2 minutes, tops.

3 Visual Feedback

I want to use column D as my automated testing column. I want it to show **Green** if the test works and **Red** (with white ink) if it fails. That way the spreadsheet should shout out at me if tests are failing.

- So I add another heading Pass/Fail and add the formula =IF(B2=C2,"pass","fail") into cell D2.
- I quickly overtype cell C2 with “i” to check that it works with a pass and it does. Then I put in the word “blah” to check that the fail works and it does to. I undo both of those and play around with

Excel's conditional formatting feature and make "pass" show in Green background and "fail" show in Red Background.



I now have a testing environment, of sorts, and a passing test.

	A	B	C	D
1	Integer	Roman	i2r	pass/fail
2	1	I	I	pass

4 Passing my 2nd test

I figure now is as good a time as any to try my second test.

- I add in the test 2 = "II" and *copy*¹ the C and D formulas from the existing row.

	A	B	C	D
1	Integer	Roman	i2r	pass/fail
2	1	I	I	pass
3	2	II	I	fail

So, now I have my 2nd test and it is failing. This is good. It's the rhythm of TDD - write a failing test, make it work using the simplest way possible, then refactor (i.e. tidy up) the code.

I reckon I know how to fix up this code.

- After a quick peek in the help files on the "if" statement, I change the code

¹ If you're unfamiliar with Excel then do make sure you copy the formulas. It's much, much, much easier and less error prone than trying to retype them.

```
Public Function i2r(i As Integer) As String
    If i = 1 Then
        i2r = "I"
    Else
        i2r = "II"
    End If
End Function
```

- Then I switch across to Excel and press F9 to recalculate.

	A	B	C	D
1	Integer	Roman	i2r	pass/fail
2	1	I	I	pass
3	2	II	II	pass

Two passing tests! I'm on a roll.

5 A few more tests, but no Refactoring.

- I take a look at the code and wonder if I can tidy it up at all (refactor) ... and I can't see anything².
- So I add in the next test: 3 = "III", it fails, I change the code so it passes, but I still can't see anything to refactor.
- I repeat this process for tests 4, 5, 6, 7, 8, adding each test - one at a time - making it work, trying to refactor.
- I start to think that this could be a long and drawn out process. I can't see anything that I can easily refactor.

```
Public Function i2r(i As Integer) As String
    If i = 1 Then
        i2r = "I"
    ElseIf i = 2 Then
        i2r = "II"
    ElseIf i = 3 Then
        i2r = "III"
    ElseIf i = 4 Then
        i2r = "IV"
    ElseIf i = 5 Then
        i2r = "V"
    ElseIf i = 6 Then
        i2r = "VI"
    ElseIf i = 7 Then
        i2r = "VII"
    ElseIf i = 8 Then
        i2r = "VIII"
    End If
End Function
```

- But then I look a little harder.

² One of my very experienced programmer friends told me that I could have refactored the code at this stage by sorting out the repeating "I"s, for example. But, to be honest, I'm not a programmer any more and I didn't feel I could do so comfortably.

6 Some Refactoring

I see a pattern with the 1, 2 and 3 and 6, 7, 8. I wonder if I can do something with the repeating "I"s.

But what to do?

When I consider the code, I see that there are two ways in which I think I can tidy up the code (refactor). First, I could turn 1, 2 and 3 into a loop. Second, I could turn 6, 7 and 8 into (5 + 1, 2 or 3). I *think* I'm clever enough to do both at the same time but then I recall the XP idea of doing things one at a time, since it's more likely to work/ less likely to break. I flip an imaginary coin and decide to turn 1, 2 and 3 into a loop.

- First, I'm not sure quite why, but I move the code for 1, 2 and 3 down to the bottom of the code.

```
Public Function i2r(i As Integer) As String

If i = 4 Then
    i2r = "IV"
ElseIf i = 5 Then
    i2r = "V"
ElseIf i = 6 Then
    i2r = "VI"
ElseIf i = 7 Then
    i2r = "VII"
ElseIf i = 8 Then
    i2r = "VIII"
End If

If i = 1 Then
    i2r = "I"
ElseIf i = 2 Then
    i2r = "II"
ElseIf i = 3 Then
    i2r = "III"
End If

End Function
```

- Just to make sure this hasn't broken anything I switch back to the spreadsheet and recalc. I probably didn't need to do this but it has been a long time since I've coded and I'm a bit nervous doing this. All of the tests still work.

	A	B	C	D
1	Integer	Roman	i2r	pass/fail
2	1	I	I	pass
3	2	II	II	pass
4	3	III	III	pass
5	4	IV	IV	pass
6	5	V	V	pass
7	6	VI	VI	pass
8	7	VII	VII	pass
9	8	VIII	VIII	pass

- Then I turn the 1,2 and 3 bits into a loop.

This is my first attempt:

```
Public Function i2r(i As Integer) As String
    If i = 4 Then
        i2r = "IV"
    ElseIf i = 5 Then
        i2r = "V"
    ElseIf i = 6 Then
        i2r = "VI"
    ElseIf i = 7 Then
        i2r = "VII"
    ElseIf i = 8 Then
        i2r = "VIII"
    End If

    i2r = ""
    While i < 3 And i > 0
        i2r = i2r + "I"
        i = i - 1
    End
End Function
```

I recalc the spreadsheet and it doesn't work. I realise this when I look at the code and notice I put a END, rather than a WEND, to close off the While loop.

So, I pop back to the spreadsheet and press F9 to rerun the tests again.

7 Ooops. I've broken lots of tests.

	A	B	C	D
1	Integer	Roman	i2r	Pass/fail
2	1	I	I	pass
3	2	II	II	pass
4	3	III		fail
5	4	IV		fail
6	5	V		fail
7	6	VI		fail
8	7	VII		fail
9	8	VIII		fail

When I look back at the code I realise that I've done two things wrong.

- First, the loop should be <= 3, not < 3. I change that, recalc the tests and 3 is fixed.

But what about 4 - 8? They're still broken. And, no wonder, the i2r = "" before the loop destroys them.

- I fix the code for the current tests by moving i2r= "" to the top of the code.

When I recalc the spreadsheet ... all of the tests work! The tests are very comforting ... a safety net.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 4 Then
    i2r = "IV"
ElseIf i = 5 Then
    i2r = "V"
ElseIf i = 6 Then
    i2r = "VI"
ElseIf i = 7 Then
    i2r = "VII"
ElseIf i = 8 Then
    i2r = "VIII"
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

So. The 2nd Refactoring. Hmm. I want to change the code so that 6, 7, and 8 are 5 + 1, 2, or 3.

I have a wee discussion with myself before making any changes and decide to comment out the 6, 7 and 8 elseif's, change the if i = 5 to if i >= 5, and subtract 5 from the i2r, and see how it works. I make these changes and break a lot of the tests. Hmm. What to do?

- I turn on the debugger. And the moment I step through the very simple code I see a blindingly obvious keying error that I could have found without the debugger if only I'd looked at the code a little harder.
- I fix my mistake. I turn off the debugger.
- I switch to Excel and recalc.
- All of the tests pass.
- I remove the code that I'd commented-out.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 4 Then
    i2r = "IV"
ElseIf i >= 5 Then
    i2r = "V"
    i = i - 5
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

I look at the code and I'm happy with it, but not thrilled, so I look to see if I could refactor it a bit more. I decided, instead, to add a few test cases to

understand the problem better. In other words, I want to do a bit more analysis and I am going to do it with the code.

- I add the test 9 = IX. It fails. I make it work by adding another ifelse, just like the 4. Can't see any way to refactor it.
- I add the test 10 = X. It fails. Again I add it as another ifelse, just like the 4 and 9.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 4 Then
    i2r = "IV"
ElseIf i = 9 Then
    i2r = "IX"
ElseIf i = 10 Then
    i2r = "X"
ElseIf i >= 5 Then
    i2r = "V"
    i = i - 5
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- As I add the test 11 = XI, I notice the i following the x and I think about the 12 and 13. This looks like it's the good old repeating "I" situation.

I'm tempted to add the 12 and 13 test cases in too, but I decided not to. Why? After all, it seems easy enough to do, but then I'd move into trying to make 3 tests work at once and although they'd probably work I suspect that it's not the intention of TDD. And besides, it is easy enough to add the 12 and 13 in afterwards.

- I make the 11 test work very easily

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 4 Then
    i2r = "IV"
ElseIf i = 9 Then
    i2r = "IX"
ElseIf i >= 10 Then
    i2r = "X"
    i = i - 10
ElseIf i >= 5 Then
    i2r = "V"
    i = i - 5
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- I look to see if I can refactor anything. I can't see anything.
- I add in the 12 = XII test case. It works straight away. As does the 13 = XIII.
- I add in the 14 = XIV test case. Now the easiest way to make this test work is to put in an if i = 14 and I do this. But when I go to refactor I realise that it's really just 10 + 4, so I move the if i =4 out of the string of if's and move it after the 5 and between the 3 , 2 and 1 code. It should just trickle through. My tests still all run after making the change.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 9 Then
    i2r = "IX"
ElseIf i >= 10 Then
    i2r = "X"
    i = i - 10
ElseIf i >= 5 Then
    i2r = "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- I look for ways to refactor but I (with my limited coding ability) can't see any that I can do with confidence.

- I add the 15 = XV test case. I'm starting to see a pattern here and I change the code easily.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 9 Then
    i2r = "IX"
ElseIf i >= 10 Then
    i2r = "X"
    i = i - 10
End If

If i >= 5 Then
    i2r = "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

But 15 doesn't work and the test for 9 breaks.

	A	B	C	D
1	Integer	Roman	i2r	pass/fail
2	1	I	I	pass
3	2	II	II	pass
4	3	III	III	pass
5	4	IV	IV	pass
6	5	V	V	pass
7	6	VI	VI	pass
8	7	VII	VII	pass
9	8	VIII	VIII	pass
10	9	IX	VIV	fail
11	10	X	X	pass
12	11	XI	XI	pass
13	12	XII	XII	pass
14	13	XIII	XIII	pass
15	14	XIV	XIV	pass
16	15	XV	V	fail

I suspect if you've been following along in Excel then you've discovered you too make lots of little mistakes as you go. At least I hope so and that it isn't just me. It is one of the nice things about all those tests ... when you break something by making a simple mistake, the tests tell you, you fix it quickly and cheaply, and move on.

- When I see that the 9 test case is also broken it is immediately obvious what I've done wrong.

- I fix it quickly. The tests work.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i = 9 Then
    i2r = "IX"
    i = i - 9
End If

If i >= 10 Then
    i2r = i2r + "X"
    i = i - 10
End If

If i >= 5 Then
    i2r = i2r + "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- Then I do a bit of tidying up - moving things around a little.
- For each small change I make, I recalc the tests.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i >= 10 Then
    i2r = i2r + "X"
    i = i - 10
End If

If i = 9 Then
    i2r = i2r + "IX"
    i = i - 9
End If

If i >= 5 Then
    i2r = i2r + "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
End If

While i <= 3 And i > 0
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- Then I do a bit more tidying up.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i >= 10 Then
    i2r = i2r + "X"
    i = i - 10
End If

If i = 9 Then
    i2r = i2r + "IX"
    i = i - 9
End If

If i >= 5 Then
    i2r = i2r + "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
    i = i - 4
End If

While i >= 1
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

- Then I look at the code and I see the obvious pattern. It jumps out at me now. I can create new function for each of the numbers.
- I try, but fail, to add the new function -out of laziness and lack of imagination I give up and decide to add more test cases first.
- I add a test case for 16 =XVI and then I get enthusiastic and add the tests into the spreadsheets for 17 through to 19. I think they should work straight away and they do.
- I add 20 = XX and it doesn't work immediately.
- I look at the code and quickly change the IF i >= 10 to a while loop.

Hmmmm, I think. I reckon that these tests should now work all the way to 50.

- So I add in the integer values from 21 - 30 and instead of filling in the roman numerals, I cheat a little and copy down the i2r() function and eyeball them to check that they're right. They all are so I copy the values from my function into the test data.
- I choose a few other numbers up to 50, which I figure will fail with the current code. But, when I type in 46 I realise that the 40's are a special case like 4 and 9, and later on 90, 400 and 900. So I check in Google³ what the symbol for 40 is ... it's XL. I add in the test case for 40 and make it work.
- I add in a test case for 49 and it works. I'm confident that the rest of the 40's will work too.

³ <http://www.google.co.uk/search?q=40+in+roman+numerals>

- I add in a test case for 50 = l and as expected it doesn't work. By now, the code is trivial so I make it work.
- Then I think about some Refactoring.

The code looks like this:

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

If i >= 50 Then
    i2r = i2r + "L"
    i = i - 50
End If

If i >= 40 Then
    i2r = i2r + "XL"
    i = i - 40
End If

While i >= 10
    i2r = i2r + "X"
    i = i - 10
Wend

If i = 9 Then
    i2r = i2r + "IX"
    i = i - 9
End If

If i >= 5 Then
    i2r = i2r + "V"
    i = i - 5
End If

If i = 4 Then
    i2r = i2r + "IV"
    i = i - 4
End If

While i >= 1
    i2r = i2r + "I"
    i = i - 1
Wend

End Function
```

I figure that I want to create a function where I feed in each *roman numeral* and if the current value of *i* > *the decimal equivalent* of the *roman numeral* then concatenate the *roman numeral* on the end of *i2r* and subtract the *decimal equivalent* from *i*.

The only thing that is stopping me is the mixture of whiles and if's in the pattern above. The I's and X's are whiles, but the rest are if's. Hmmm, I think. After a bit of contemplation I'm pretty sure that each of the if's can simply be replaced with a while.

But, how can I be sure? The easiest way to be sure is to try it and see. I have the tests to save me if I make a mistake and I have undo to roll back if I need to. But, to be on the safe side I decide to try this from the bottom up. That is, I start with the I's.

This is what I come up with, for the I's (I couldn't think of a good function name, so m it is):

```
...
i2r = i2r + m(i, "I", 1)
'While i >= 1
'  i2r = i2r + "I"
'  i = i - 1
'Wend

End Function

Public Function m(ByRef i As Integer, ByVal RomanCharacter As String, ByVal
IntegerValue As Integer) As String
While i >= IntegerValue
    m = m + RomanCharacter
    i = i - IntegerValue
Wend
End Function
```

And it works.

- So I try it on the 4 IF, replacing the if i = 4 with i2r = i2r + m(i, "IV", 4)
- And it works.
- So I try it on the 5. It works too.
- So, I do all of the individual roman numerals. Recalculating the tests after each small change.

I end up with this:

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = ""

i2r = i2r + m(i, "L", 50)
i2r = i2r + m(i, "XL", 40)
i2r = i2r + m(i, "X", 10)
i2r = i2r + m(i, "IX", 9)
i2r = i2r + m(i, "V", 5)
i2r = i2r + m(i, "IV", 4)
i2r = i2r + m(i, "I", 1)

End Function

Public Function m(ByRef i As Integer, ByVal RomanCharacter As String, ByVal
IntegerValue As Integer) As String
While i >= IntegerValue
    m = m + RomanCharacter
    i = i - IntegerValue
Wend
End Function
```

And the tests all work afterwards.

- Then I refactor the i2r() function.

```
Public Function i2r(i As Integer) As String
Application.Volatile

i2r = m(i, "L", 50) +
    m(i, "XL", 40) +
    m(i, "X", 10) +
    m(i, "IX", 9) +
    m(i, "V", 5) +
```

```
m(i, "IV", 4) +  
m(i, "I", 1)
```

End Function

I'm confident that I understand how this function is supposed to work so I quickly add in the 90 = XC , 100 = C , 400= CD, 500 = D, 900 = CM and 1000 = M code, their test cases, and a healthy smattering of tests in between. When I get to 1000 = 'M' I realised that my laziness with the function name m() was very sloppy and I change it. The variable named i wasn't very good either... but, easy enough to fix.

8 Finally

If you've worked your way through to the end of the example then I expect that, like me, your first reaction was surprise that the process worked and that the solution turns out to be simple. My second reaction was to wonder if it would have just been easier to do a bit more analysis on the roman numeral rules up front and just code them.

The truth is ... I don't know what approach would be most efficient for the i2r() example. But I spend a lot of time in the company of TDD experts and they tell me that the i2r() example is a *great way to demonstrate TDD*, but it's not a *great example of TDD*. The power of TDD really comes out when working on bigger problems.

A friend⁴ reviewing this document commented,

“As for your conclusion...who knows if an initial analysis/design up front would have led you to the same elegant solution? Perhaps that isn't the important point of TDD. When using TDD I always find that I am far more familiar, and hence able to talk about, the eccentricities in the domain that I've uncovered through TDD and much more open to incorporating future eccentricities as they are uncovered by myself or my customer. I'm already in the change frame of mind.”

So, in conclusion, I hope that this example has given you a feel for how TDD works, that TDD can work and that TDD is worth exploring further.

⁴ I'd like to thank the large number of people who have kindly provided feedback on this document. If you have further feedback then please email me on Clarke.ching@gmail.com.