Visual Web Development

Terry Marris November 2008

11 Testing, Debugging and Validation

We have looked at sequences, selections and iterations. Now we look at testing, debugging and validation.

11.1 Purpose of Testing

The purpose of testing is to find errors. A successful test is one that discovers an error. If you had a job as a tester your job would be to find errors. No matter how much testing you do, you can never be sure to have found every error in a program. That is the nature of testing.

Quality assurance is demonstrating that programs function according to requirements. But there is no guarantee that requirements are unambiguous, correct, complete, or that they have been fully understood.

The mathematics of sets and logic - for example, The Z Notation - can improve the specification of program requirements. Check out http://vl.zuser.org/ for further details.

Assuming that every user is a tester has contributed to the downfall of many great companies and not a few fatalities. Check out http://catless.ncl.ac.uk/risks for some horror stories.

Programming for exceptions is perhaps the most important part of programming.

11.2 Dry Runs

We record the changing values of variables and Boolean expressions as we work through a piece of programming line by line.

For example:

I
```
Dim a, b, m, x As Integer
x = 4
m = 5
b = 8
a = 1
x = (a * x + b) Mod m
```

Create column headings, one for each variable. Progress through the code line-by-line writing down the value of each variable.

x	m	b	a
4	5	8	1
(1 x 4 + 8) Mod 5			
= 12 Mod 5			
= 2			

II
```
Dim i, s, n As Integer
n = 5
s = 0
i = 1
While i <= n
  s = i * i
  i = i + 1
End While
```

Create column headings for each variable and Boolean expression. Progress through the code line by line, looping as required.

n	s	i	i <= n
5	0	1	True
	1	2	True
	4	3	True
	9	4	True
	16	5	True
	25	6	False

III
```
Dim i
For i = 1 to 5
    If i Mod 2 = 0
        text.Output.Text = i
    End If
Next
```

Create columns for each variable and Boolean expression. Remember that i ranges from 1 up to 5 inclusive.

i	i <= 5	i Mod 2	i Mod 2 = 0?	txtOutput.Text
1	True	1	False	
2	True	0	True	2
3	True	1	False	
4	True	0	True	4
5	True	1	False	
6	False			

Dry runs become unmanageable for large amounts of programming code. An alternative is to use the debugging facilities provided by the programming environment.

11.3 Exercises

Dry run each of the following program code fragments and determine what each does.

1.
```
Dim a, b, t As Integer
a = 2
b = 3
t = a
a = b
b = t
```

2. Dim i, m, n, c As Integer (dry run three times for each m: m = 49, 50 and 51)
```
p = 50
n = 3
c = 0
i = 0
While i < n
  i = i + 1
  m = Convert.ToInt32(txtInput.Text)
  If m >=p
      c = c + 1
  End If
End While
txtOutput.Text = c
```

3. Dim i, n, a, s As Integer (dry run for a = 3, 5 and 7)
```
n = 3
i = 0
s = 0
While i < n
  i = i + 1
  a = Convert.ToInt32(txtInput.Text)
  s = s + a
End While
txtOutput.Text = s
```

4.
```
Dim n, f, i As Integer
n = 5
f = 1
For i = 1 to n
  f = i * f
Next
txtOutput.Text = f
```

5.
```
Dim a, b, c, i, n As Integer
a = 0
b = 1
c = 0
i = 2
n = 7
While i < n
  i = i + 1
  c = a + b
  a = b
  b = c
  txtOutput.Text = a
End While
```

6.
```
 Dim m, e, g, h As Double
  m = 16.0
  e = 0.01
  h = m / 2.0
  Do
     g = h
     h = (g + m / g) / 2              [ could g ever become zero? ]
  Loop Until Abs(g - h) < e
  txtOutput.Text = h
```

7.
```
Dim x, n, p, s As Integer
x = 3
p = 1
s = x
While n > 0
  If (n Mod 2) = 1 Then
     p = p * s
     n = n \ 2
     s = s * s
End While
txtOutput.Text = p
```

11.4 Test Data

We attempt to expose errors in our programs by deliberately and systematically choosing data values for testing.

We check every path through the code. We check with normal and extreme data values. We check boundary points because experience has shown that errors are likely to be found nearby.

Boundary points occur in selection and iteration statements.

```
If age >= 18 Then
   status = "OK"
Else
   status = "Too young"
End If
```

Here, the boundary point is *age >= 18*. So, for test data, we would choose age values of 17, 18 and 19 for testing - just under, exactly on and just over the boundary.

In the loop shown below, the boundary point is *i <= n*.

```
Dim i, s, n As Integer
n = 5
s = 0
i = 1
While i <= n
   s = i * i
   i = i + 1
End While
```

We choose values for n so that the loop is executed:

- zero times
- once
- many times

For example, you would have three sets of test data for when n = 0, n = 1 and n = 2.

A boundary occurs whenever a small change in value causes a large change in program behaviour.

11.5 Exercises

1. A carbon monoxide (CO) detector issues warnings according to the level of carbon monoxide detected. (ppm stands for parts per million.)

Reading (ppm)	**Report**
0-9	Normal
10-35	Marginal
36-99	Excessive
100-200	Dangerous
>200	Fatal

Devise a set of appropriate test data.

2. Devise a set of appropriate test data for the code fragment shown below.

```
Dim i, n, a, s As Integer
n = 3
i = 0
s = 0
While i < n
  i = i + 1
  a = Convert.ToInt32(txtInput.Text)
  s = s + a
End While
txtOutput.Text = s
```

11.6 Test Plans

A test plan usually has four columns.

Test #	Test Data	Reason	Expected Result
1 2 3	examMark = 39 examMark = 40 examMark = 41	fail/pass boundary	fail pass pass

Test data is the data you have chosen to test the program e.g. examMark = 39, . Reason is why you chose the test data e.g. fail/pass boundary. The expected result is the predicted result; it is vital because otherwise, how would you know whether your program was behaving correctly?

If your expected result differs from your actual result it could be because your programming logic was wrong, or your expectation was wrong, or both. You either ***acknowledge*** the difference and do nothing about resolving it - it becomes a known error, or you fix it and re-test.

11.7 Test Logs

A test log refers to an actual test run; this could be a screen dump complete with time, date and comments written by hand.

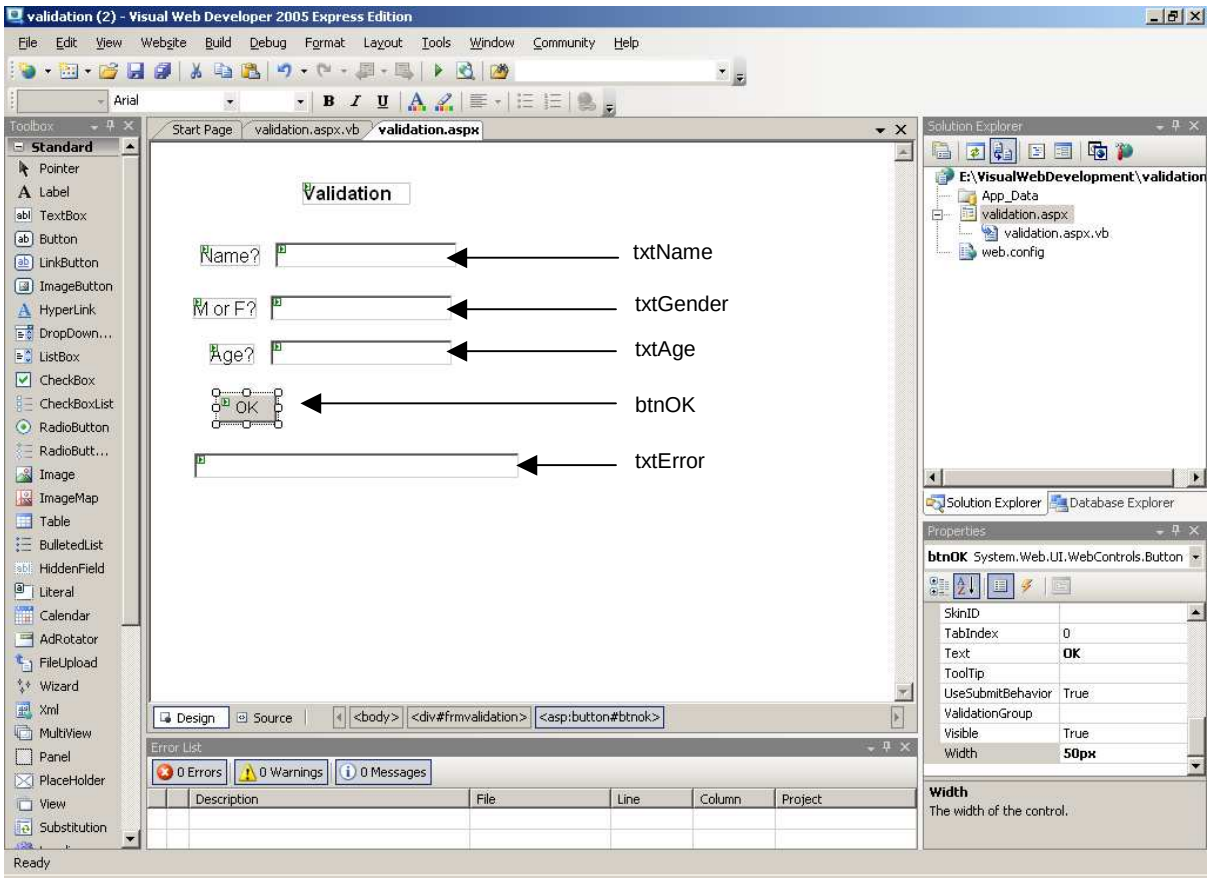
11.8 Debugging

Program development environments usually provide facilities for inspecting the values of variables as a program executes, and for halting program execution at chosen break points. You could investigate the debugging facilities provided by ASP.Net.

However, it is often easier for the programmer to create his or her own diagnostics, displaying the values of variables as required. The diagnostic code can be commented out when finished with.

11.9 Validation

Validation is ensuring that data values fall between pre-defined limits. For example, a person's age *must* be between 0 and 120, a person's gender *must* be either m or f, a person's name *must* have at least two characters.



The code under the OK button is:

```
Protected Sub btnOK_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnOK.Click
    Dim strName As String = ""
    Dim chrGender As Char = ""
    Dim intAge As Integer = 0
    Try
        strName = txtName.Text
        chrGender = Convert.ToChar(txtGender.Text)
        chrGender = Char.ToLower(chrGender)
        intAge = Convert.ToInt32(txtAge.Text)

        If strName.Length() < 2 Then
            Throw New ApplicationException()
        End If
        If chrGender <> "m" And chrGender <> "f" Then
            Throw New ApplicationException()
        End If
        If intAge < 0 Or intAge > 120 Then
            Throw New ApplicationException()
        End If

   Catch ex As ApplicationException When strName.Length() < 2
        txtError.Text = "Name must contain 2 or more characters"
Catch ex As ApplicationException When chrGender <> "m" And chrGender <> "f"
        txtError.Text = "Gender must be either m or f"
    Catch ex As ApplicationException When intAge < 0 Or intAge > 120
        txtError.Text = "Age must be between 0 and 120"
    Catch ex As FormatException
        txtError.Text = "Age must be numeric"
    End Try
End Sub
```

declares and initialises variables

gets input and converts gender to lower case and age to integer

catches the input errors - when name is too short,

when gender is neither m nor f,

when age is out of range

deals with the input errors

Using Try ... Catch ... to deal with errors has its critics and supporters. Critics say it leads to large overheads that slow program execution, solved by using simple selection statements and a function to process the error messages. Supporters say it is elegant and safe, leads to simpler programming code, and modern computers have sufficient memory and processing power to cope.

11.10 Exercises

Exhaustively test the program shown above that gets and validates name, gender and age from the user. Correct any errors you may find.

11.11 Conclusion

We have seen the importance of testing and looked some testing strategies. We have touched on debugging and looked at validation. Next, we look at state and scope.