

Teaching Visual Basic.Net: A Student Oriented Success Method

Claude Simpson

University of Texas Pan American
College of Business
CIS/QUMT MAGC 3.314
Edinburg, Texas 78541
956-381-2843 956-381-3367
csimpson@panam.edu

Lester Rydl

University of Texas Pan American
College of Business
CIS/QUMT MAGC 3.306
Edinburg, Texas 78541
956-381-3388 956-381-3367
lrydl@panam.edu

Michael Crews

University of Texas Pan American
College of Business
CIS/QUMT MAGC 3.330
Edinburg, Texas 78541
956-381-3350 956-381-3367
rmcrews@panam.edu

ABSTRACT

This paper presents an approach that is different from the common approach currently used in most texts that cover Visual Basic.Net. The approach that we use was developed to counter two major concerns:

- 1. that students really haven't learned how to program and have no idea what goes on behind the objects that are placed on a form and*
- 2. that the Visual Basic.Net language has evolved away from a language that is useful for teaching beginning programmers.*

INTRODUCTION

BASIC is an acronym that stands for Beginners All-purpose Symbolic Instruction Code. The contention of the authors of this paper is that without a radical modification of the general approach used to teach beginning programmers the VB.Net language, there are considerable problems in the learning process for new students who have never programmed.

The primary problem that we have found in teaching this language is that students can learn to design a great looking form; place all sorts of controls on the form; make it look very nice,

follow the code examples in the textbook, and still have no idea how the code achieves the desired output in the program. In the following exposition we suggest a very different approach to teaching the VB.Net language.

DESIGN FIRST

The first step in our methodology is that the student is given a problem such as:

Prompt an operator for and receive two integers. Add the two integers together. Subtract the second integer from the first integer. Multiply the first integer by the second integer. If the second integer is not zero divide the first integer by the second integer. Display the two integers, their sum, difference, product and quotient.

We will use the seven step process suggested by Professor Lesley Ann Robertson in her book, *Simple Program Design* (Robertson, 2004). The first step is to understand the problem in terms of outputs, inputs and processes. Please note that here and throughout this paper we use output, input, and process. We have found that if we focus on output first, instead of input, the program tends to become more user oriented.

Professor Robertson suggests that after carefully reading the problem, highlight the nouns and appropriate adjectives and the verbs. This is shown in the following paragraph where we have underlined the verbs and highlighted the appropriate nouns and adjectives.

Prompt an operator for and receive two integers. Add the two integers together. Subtract the second integer from the first integer. Multiply the first integer by the second integer. If the second integer is not zero divide the first integer by the second integer. Display the two integers, their sum, difference, product and quotient.

The next step is to develop a defining algorithm. We differ a little from Professor Robertson at this point; she uses a table to create the algorithm; we use pseudocode in Microsoft's Notepad. The results are essentially the same. We carefully point out to students that the verbs describe whether the nouns involved are output, input, or processes. For example, receive can be only for input, display only for output and add only for processing. If the verb is, for example, an input operation the verb will point to the noun(s) that are input. In the above example the verb *receive* is pointing to *two integers*; the verb *display* is pointing to *two integers, sum, difference, product, and quotient*.

The defining algorithm for the problem is listed below:

Lab 3-1 defining algorithm

outputs

- integer_1
- integer_2
- sum
- difference
- product
- quotient

```

inputs
    integer_1
    integer_2
work area
    string_in
    toggle_switch[rl]
processes
    prompt for input
    receive integer_1 integer_2
    test value of integer_2
    calculate sum, difference, product, quotient
    display integer_1 integer_2 sum difference product quotient

end

```

The next step is to develop the solution algorithm, a test data set, expected results, and a desk check of the algorithm using the test data set. The solution algorithm that we developed is listed below.

Lab 3-1 solution algorithm

```

calculate program
toggle_switch = 0
prompt for integer_1
receive integer_1
prompt for integer_2
receive integer_2
sum = integer_1 + integer_2
difference = integer_1 - integer_2
product = integer_1 * integer_2
if integer_2 = 0 then
    display "division by zero error"
    toggle_switch = 1
else
    quotient = integer_1 / integer_2
endif

if toggle_switch = 0 then
    display integer_1 integer_2 sum difference product quotient
else
    display integer_1 integer_2 sum difference product
end if
toggle_switch = 0
end

```

Test data set

	Integer_1	Integer_2
Set 1	4	2
Set 2	8	0

Expected results

	Sum	Difference	Product	Quotient
Set 1	6	2	8	2
Set 2	8	8	0	Divide error

The next step is to desk check. We only show the desk check for one test data set in this paper. Also, Professor Robertson uses a table to desk check. We just write or type the results into a copy of the solution algorithm.

In the student's work all test data sets should be executed against the algorithm.

Lab 3-1 solution algorithm

calculate program

toggle_switch = 0 (assignment of value)

prompt for integer_1 (*Enter Integer 1*)

receive integer_1 (*Integer_1 = 4*)

prompt for integer_2 (*Enter Integer 2*)

receive integer_2 (*Integer_2 = 2*)

sum = integer_1 + integer_2 (*sum = 6; 6 = 4 + 2*)

difference = integer_1 - integer_2 (*difference = 2; 2 = 4-2*)

product = integer_1 * integer_2 (*product = 8; 8 = 2 * 4*)

if integer_2 = 0 then (*not executed*)

display "division by zero error"

toggle_switch = 1

else

quotient = integer_1 / integer_2 (*quotient = 2; 2 = 4 / 2*)

endif

if toggle_switch = 1 then

display integer_1 integer_2 sum difference product (*all results are displayed except quotient*)

else

display integer_1 integer_2 sum difference product quotient (*all results are displayed*)

end if

```
toggle_switch = 0    (assignment of value)
```

```
end
```

CODE LAST

The next step in the process is to code the program in the language of choice. We begin with VB.Net *console applications* instead of Windows Application because this is where we prepare students for what is really going on behind the scenes in the windows applications.

The defining algorithm is used to define the data that are used in the program and the solution algorithm is used for the program logic that is written in the program module. The logic from the console application is listed below:

Module Module1

```
'John Q. Student  
'lab 3-1
```

```
'outputs  
Dim sum As Integer  
Dim difference As Integer  
Dim product As Integer  
Dim quotient As Integer
```

```
'inputs  
Dim integer_1 As Integer  
Dim integer_2 As Integer
```

```
'work area  
Dim string_in As String  
Dim toggle_switch As Integer = 0
```

```
sub main()  
    Console.WriteLine("Enter integer 1:") 'prompt for integer 1  
    string_in = Console.ReadLine()      'get the first integer; it comes in the program as a  
string  
    integer_1 = Convert.ToInt16(string_in) 'convert string_in to integer data and store it in  
integer_1  
  
    Console.WriteLine("Enter integer 2") 'Follow the process used above to get the second  

```

```

difference = integer_1 - integer_2    'compute difference
product = integer_1 * integer_2      'compute product

If integer_2 = 0 Then                'avoid divide by zero
    Console.WriteLine("divide by zero error") 'error message; we could have used
msgbox command here
    toggle_switch = 1                ' set the toggle_switch to not print output
Else
    quotient = integer_1 / integer_2 'compute quotient
End If

If toggle_switch = 0 Then
    Console.WriteLine("integer 1 = {0} integer 2 = {1} sum = {2} diff = {3} product = {4}
quot = {5} ", _
        integer_1, integer_2, sum, difference, product, quotient)
Else
    Console.WriteLine("integer 1 = {0} integer 2 = {1} sum = {2} diff = {3} product = {4}
", _
        integer_1, integer_2, sum, difference, product)
End If
        toggle_switch = 1            ' set the toggle_switch to not print output

Console.ReadLine()                  'this command stops the screen

End Sub

End Module

```

One of the test sets captured with the Print/Scrn key and then pasted into this document is shown below:

The screenshot shows a console window titled "C:\Documents and Settings\csimpson\My Documents\Visual Studio Projects\ConsoleApplicat...". The output text is as follows:

```

Enter integer 1:
8
Enter integer 2:
0
divide by zero error
integer 1 = 8 integer 2 = 0 sum = 8 diff = 8 product = 0

```

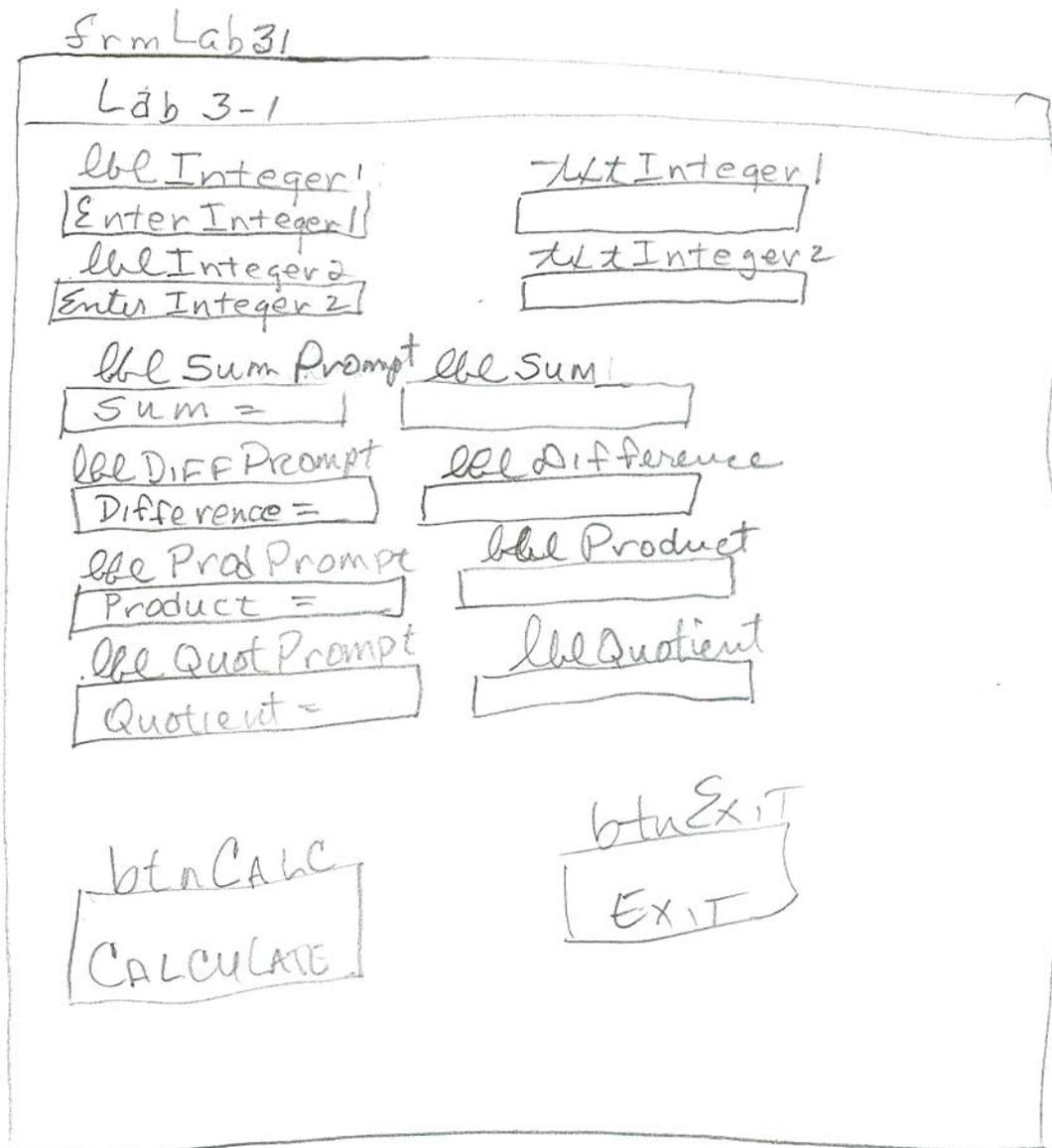
The test results are compared to the expected results and in all cases the expected results and the program results are identical. The test set should include all possibilities for the logic. The test sets that we used did not include divide by zero and should have been included.

The rest of the paper will address how the same problem is solved in a Windows Visual Studio.Net (VB) application.

THE WINDOWS SOLUTION

The steps used in the development of the Windows solution are the same as the steps used above except that we insert a drawing of the user-interface so that we can determine, to a degree, what the form is going to look like and the names that the objects on the form will take on.

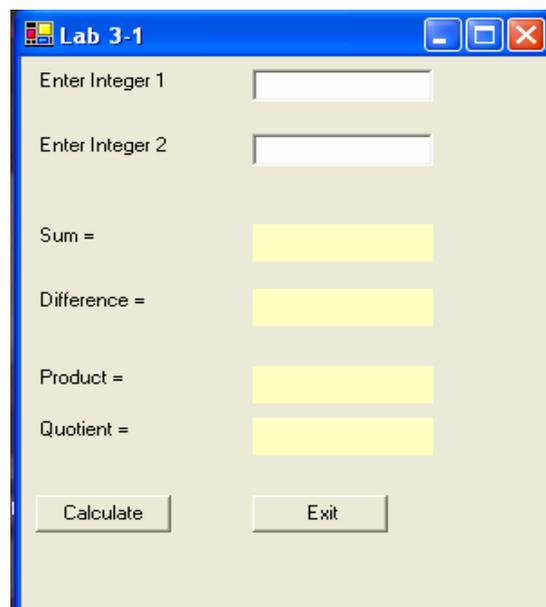
A Penciled Diagram of the Graphical User Interface (GUI)



Some faculty prefer Task Object Event (TOE) charts. This is fine; use whatever serves your students best. We prefer a graphical diagram as indicated above with the objects labeled and some idea of how the form should look. True, this does not look all that great but it does give the student an idea of what s/he wants the form to look like.

Drawing the GUI Form in VB.Net

Once the diagram is drawn; all the student has to do is use the toolbox to place the objects on the form and name each object's properties. We have captured the form with the objects we placed on it and have pasted it into this document.



As can be seen here the form is somewhat close to the penciled GUI interface. If we were to print the object name list you could see that the object's properties have been named in accordance with the penciled diagram.

The VB.Net Code

The Visual BASIC code for this problem is listed below.

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExit.Click
    Me.Close()
End Sub
Private Sub btnCalc_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnCalc.Click
    'please do not worry, at this point, about global or local variables.
    'outputs
    'please note that the outputs are defined on the form as labels.
```

```
'inputs
Dim integer_1 As Integer
Dim integer_2 As Integer
```

```
integer_1 = txtInteger1.Text    'this command gets the text value out of the text box
integer_2 = txtInteger2.Text    'This command gets the integer 2 value
```

```
lblSum.Text = integer_1 + integer_2    'add the integers and move them to the appropriate
label
```

```
lblDifference.Text = integer_1 - integer_2    'subtract integer 1 from integer2 and put in label
```

```
lblProduct.Text = integer_1 * integer_2    'multiply the values and put in appropriate label
```

```
'before you can divide you must check for divide by zero which is not possible.
```

```
If integer_2 = 0 Then
```

```
    lblQuotient.Text = "Error"    'error for divide by zero
```

```
Else
```

```
    lblQuotient.Text = integer_1 / integer_2
```

```
End If
```

```
End Sub
```

```
Private Sub txtInteger1_GotFocus(ByVal sender As Object, ByVal e As System.EventArgs)
Handles txtInteger1.GotFocus
```

```
    lblSum.Text = ""
```

```
    lblDifference.Text = ""
```

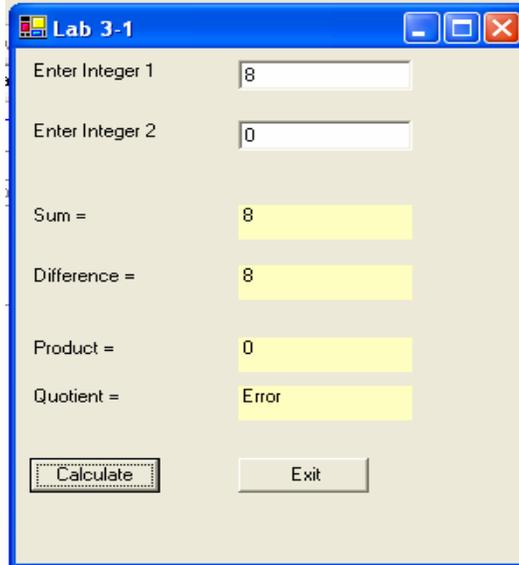
```
    lblProduct.Text = ""
```

```
    lblQuotient.Text = ""
```

```
End Sub
```

```
End Class
```

This is a very rough program and it is written to parallel the monitor program above. There are more things that should have and would have been done to make this a complete program. The objective here is to show students how to create a form and put the appropriate code with, in this program, the command buttons. If you run this program you will see that the outputs received here are the same as those obtained in the Console Application program.



SUMMARY AND CONCLUSIONS

We believe that this approach gives the student a better idea of what she or he is doing than getting hung up in most textbook's chapter 3 which usually is covering all the objects available in VB.Net and getting them on the form.

REFERENCES

Robertson, L. (2004). Simple Program Design: a Step by Step Approach. (Boston: Thomson)

The books integrate perfectly: Visual Basic.NET students who want an even stronger understanding of the .NET platform can refer to the companion intermediate-to-advanced-level title, *Application Development Using Visual Basic .NET. Look Inside*. Sample chapter is available for download in PDF format.Â *Object-oriented VB.NET programming design, in detail*â€”Explains how to design effective object-oriented Visual Basic.NET programs for distributed environments. Helps students gain the full power of VB.NET â€” and maximize reuse and efficiency â€” whether or not they have previously used true object-oriented languages. *Detailed Windows Forms coverage*â€”Demonstrates how to use Windows Forms to build robust, next-generation Windows user interfaces. VB.NET has a much richer set of libraries. Once you learn VB.NET you can easily switch over to C# as the differences between the two language lie mostly in the their syntax. As a student you want to be exposed as many different language type as possible. Focus on picking a representative from a class rather sheer quantity. For example it would be better to learn LISP then another C like language if you already know C++ or Java.Â The exposure to the methods that different languages use will greatly broaden your ability to use any language. Finally remember there is a differene between a framework and the language.Today not only there is a language to be learned but many come with a framework of useful function and classes. share|improve this answer.