Company Overview:
Rohde & Associates, Inc. is an architectural firm with a staff of six employees. They have been in business since 1997 and have produced a wide array of projects throughout the state of Alaska. The focus is primarily on commercial grade design/build and bid projects but the company is fully qualified to produce residential construction documents. Their clients vary greatly from private owners, to state and government contracts. The latter makes up 90% of the work. Notable structures designed by Rohde & Associates, Inc. since 1997 are The newly remodeled East High School, The expanded wing of the Anchorage Senior Center, Ding How Mongolian BBQ, and the adjacent Samurai Sushi. Outside of Anchorage, The expansion to the Kodiak Senior Center, The new animal control shelter in Valdez, multiple restaurants in Fairbanks, and multiple tribal council buildings scattered in the northern bush villages. The company has grown since 1997 from a single employee to its current staff producing over $1,000,000.00 gross a year in contracts.

Project Overview:
Rohde & Associates, Inc., like the majority in the construction design industry today uses CAD (Computer Assisted Design) to produce construction documents. AutoCAD™ produced by Autodesk™ is their primary tool of production. Currently, all employees in the company have AutoCAD installed and running on their station. As is the case with a majority of things in life, redundancy is rampant within the designs of many projects and across multiple projects. For example, steel I beams are standard across the entire industry as specified by the AISC (American Institute of Steel Construction). A W12X14 is identical across designs, regardless of the type of structure. The only difference is the type of anchor bolts holding it in the concrete and the size of the base plate it sits on. A VBA routine or LISP routine would be an excellent choice for automating this otherwise redundant, and time consuming task of drawing a plan view of a W12X14 with 4 anchor bolts and a 12X16 base plate. Another example would be routine tasks within AutoCAD. Plotting, or printing as other applications refer to it. Plotting is very redundant and on the larger projects, can consume an entire day in itself. Another good routine would be one that takes the page setup from one drawing layout and repeats the plot procedure for every drawing sheet without the need of user intervention. The development of these routines was, in part, my assignment. Also my assignment, to convert the already existing routines the company has, written in an antiquated language LISP, into a modern language like VBA or C++. Finally, after completing the conversions of existing routines and developing a few new, needed routines, wrap them into one easy to install application that can be easily distributed as needed.

Planning Process:
The planning process what very cut and dry for this assignment, the company stated the process by which I should advance. First, convert the existing routines
into a language that is more user friendly and future development friendly. The only planning required here was internal planning on my part determining what routines to analyze first and start coding in VBA. The language to use was left for me to decide, but there are very few logical choices. Once an order of precedence was determined I proceeded to convert each one individually. Determining which to convert first was decided by which script appeared to require the least amount of lines of code. The logic to this was the learning curve I needed to overcome. I understood Visual Basic, but Visual Basic for AutoCAD, I was unfamiliar with. The application specific method calls and how objects were containerized was foreign to me. Starting easy and building up helped me learn as I progressed making the larger routines easier to design.

**Detailed Analysis and Design**

VBA is a very direct and easy to use, block code implemented design solution. The understanding by the company was that the code was going to be written so methods could be added to the routines later to improve their performance. This idea was carried through into the install application that was to be written in C++.

**Description of Implementation**

I started with two widely used and very popular routines that dramatically increase productivity within the company, “fixblock” and “stairs”. Stairs is a very simple routine that draws a staircase. This routine is very nice because of the simple fact that it eliminates the math. Instead of the CAD operator having to manually figure out how many stair risers are needed to get from one floor to the next, that meet code, you run the routine, answer the questions as the routine progresses and when finished, you have a legal staircase drawn. Fixblock is a utility that fixes improper drawings. What that means is, as firms exchange drawings amongst each other for large projects, Rohde & Associates receives drawings with improper line weights and line types for particular objects, say a kitchen sink for the purposes of this example. The sink is on a layer that makes it print so dark, it prints out as a black blotch on our plotter. The only way to fix this without this routine is to manually go into the drawing that contains that sink, redraw the sink, re-block it together, save it, and reload it into the drawing. This is extremely time consuming, especially if you have forty sinks in this particular building. Fixblock, when used allows you to select the object, or block, within the drawing, then automatically enters the drawing containing the sink, changes the block to the proper line weights, re-blocks, saves and reloads the drawing. Both of these particular routines were converted from LISP to VBA. The design was to be object oriented and utilized existing encapsulation of VBA objects, together with custom AutoCAD method calls to objects. ObjectARX and C++ was implemented with a windows based .dll template that is once again object oriented.
Product Discussion
Currently, the project is still in a state of development. All of our commonly used routines are converted to VBA for ease of use amongst the users. Developing these same routines in ObjectARX and C++ is currently in a state of disarray. The packaging software is InstallShield but currently only installs one routine successfully. A major setback in the design was the idea to upgrade to AutoCAD 2006 in the middle of this assignment. There are quite a few method calls that are changed for ObjectARX 2006 vs. ObjectARX 2004 which was the previous version of AutoCAD that was installed.

Project Conclusion
To conclude, VBA development for AutoCAD will always be an on going task. This particular assignment given to me developed very smoothly within the VBA environment. Using C++ to develop these same routines is much more cumbersome. Implementing a company wide upgrade to AutoCAD 2006 did not help with my C++ development problems. I have now switched over completely to development of routines within C++.
Code Examples

- This is the base plate routine written in VBA, My comments are mostly internal and helpful to myself as not many other people were going to understand this code anyways.

Private Sub CommandButton1_Click()

Dim padx, pady, platex, platey As Double
Dim padxhalf, padyhalf, platexhalf, plateyhalf As Double
Dim abdia, abcount, abdistx, abdisty As Double

pady = theform.TextBox1.Text
pady = theform.TextBox2.Text
platex = theform.TextBox3.Text
platey = theform.TextBox4.Text
abdia = theform.TextBox5.Text
abcount = theform.TextBox6.Text
abdistx = theform.TextBox7.Text
abdisty = theform.TextBox8.Text

padyhalf = pady / 2
padyhalf = pady / 2
platexhalf = platex / 2
plateyhalf = platey / 2

's the following draws the pad, nothing to magical here

Dim pad As Acad.LWPolyline
Dim padpoints(0 To 9) As Double

padpoints(0) = 0: padpoints(1) = 0
padpoints(2) = padx: padpoints(3) = 0
padpoints(4) = padx: padpoints(5) = pady
padpoints(6) = 0: padpoints(7) = pady
padpoints(8) = 0: padpoints(9) = 0

Set pad = ThisDrawing.ModelSpace.AddLightWeightPolyline(padpoints)

's the following draws the baseplate, it's pretty straight forward

Dim plate As Acad.LWPolyline
Dim platepoints(0 To 9) As Double

platepoints(0) = (padyhalf - platexhalf): platepoints(1) = (padyhalf - plateyhalf)
platepoints(2) = (padyhalf + platexhalf): platepoints(3) = (padyhalf - plateyhalf)
platepoints(4) = (padyhalf + platexhalf): platepoints(5) = (padyhalf + plateyhalf)
platepoints(6) = (padyhalf - platexhalf): platepoints(7) = (padyhalf + plateyhalf)
platepoints(8) = (padyhalf - platexhalf): platepoints(9) = (padyhalf - plateyhalf)

Set plate = ThisDrawing.ModelSpace.AddLightWeightPolyline(platepoints)
' the following mess draws the column
' if you change the math inside the array, and it's not working, it's probably the parenthesis
' for some odd reason, VBA is very picky about when and how you use parenthesis

Dim column As AcadLWPolyline
Dim columnpoints(0 To 33) As Double

columnpoints(0) = (padxhalf - platehalf + 0.25): columnpoints(1) = padyhalf - plateyhalf
columnpoints(2) = (padxhalf + platehalf - 0.25): columnpoints(3) = padyhalf - plateyhalf
columnpoints(4) = (padxhalf + platehalf) - 0.25: columnpoints(5) = (padyhalf - plateyhalf) + 0.25

columnpoints(6) = (padxhalf + 0.375): columnpoints(7) = (padyhalf - plateyhalf) + 0.25
columnpoints(8) = (padxhalf + 0.125): columnpoints(9) = (padyhalf - plateyhalf) + 0.5
columnpoints(10) = (padxhalf + 0.125): columnpoints(11) = (padyhalf + plateyhalf) - 0.5
columnpoints(12) = (padxhalf + 0.375): columnpoints(13) = (padyhalf + plateyhalf) - 0.25
columnpoints(14) = (padxhalf + platehalf) - 0.25: columnpoints(15) = (padyhalf + plateyhalf) - 0.25

Set column = ThisDrawing.ModelSpace.AddLightWeightPolyline(columnpoints)

'this sets the fillets for the column, it's a method call inside the polyline object

column.SetBulge 3, -0.375
column.SetBulge 5, -0.375
column.SetBulge 11, -0.375
column.SetBulge 13, -0.375

column.Update

' This section hatches the column, it uses the columnpoints array for the outer loop hatch boundary

If CheckBox1.Value = -1 Then

Dim columnhatch As AcadHatch
Dim patternName As String
Dim PatternType As Long
Dim bAssociativity As Boolean

patternName = "ANSI31"
PatternType = acHatchPatternTypePreDefined '0
bAssociativity = True
Set columnhatch = ThisDrawing.ModelSpace.AddHatch(PatternType, patternName, bAssociativity)

Dim outerLoop(0 To 0) As AcadEntity
Set outerLoop(0) = ThisDrawing.ModelSpace.AddLightWeightPolyline(columnpoints)
columnhatch.AppendOuterLoop (outerLoop)
columnhatch.Evaluate
End If

' this section draws the anchor bolt holes, it's a simple loop, that draws 2 holes at a time
' it negates the center value for the second hole and calls the method again

Dim anchorbolthole As AcadCircle
Dim count As Integer
Dim aboffset As Double
count = 1
aboffset = abdisty

Dim abcenterline1, abcenterline2 As AcadLine
Dim startPoint(0 To 2) As Double
Dim endPoint(0 To 2) As Double

For count = 1 To (abcount / 2) Step 1

    Dim centerPoint(0 To 2) As Double
    Dim radius As Double

    radius = abdia / 2

    ' the following draws the anchor bolt holes
    centerPoint(0) = padxhalf + abdistx: centerPoint(1) = (padyhalf - plateyhalf) + abdisty:
    centerPoint(2) = 0#
    Set anchorbolthole = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)
    centerPoint(0) = padxhalf - abdistx: centerPoint(1) = (padyhalf - plateyhalf) + abdisty:
    centerPoint(2) = 0#
    Set anchorbolthole = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)

    ' the following draws the center lines
    startPoint(0) = padxhalf + abdistx - 1.5: startPoint(1) = (padyhalf - plateyhalf) + abdisty:
    startPoint(2) = 0#
    endPoint(0) = padxhalf + abdistx + 1.5: endPoint(1) = (padyhalf - plateyhalf) + abdisty:
    endPoint(2) = 0#
    Set abcenterline1 = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)
startPoint(0) = padxhalf + abdistx: startPoint(1) = (padyhalf - plateyhalf) + abdisty - 1.5:
startPoint(2) = 0#
endPoint(0) = padxhalf + abdistx: endPoint(1) = (padyhalf - plateyhalf) + abdisty + 1.5:
endPoint(2) = 0#
Set abcenterline2 = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)

startPoint(0) = padxhalf - abdistx + 1.5: startPoint(1) = (padyhalf - plateyhalf) + abdisty:
startPoint(2) = 0#
endPoint(0) = padxhalf - abdistx - 1.5: endPoint(1) = (padyhalf - plateyhalf) + abdisty:
endPoint(2) = 0#
Set abcenterline1 = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)

startPoint(0) = padxhalf - abdistx: startPoint(1) = (padyhalf - plateyhalf) + abdisty - 1.5:
startPoint(2) = 0#
endPoint(0) = padxhalf - abdistx: endPoint(1) = (padyhalf - plateyhalf) + abdisty + 1.5:
endPoint(2) = 0#
Set abcenterline2 = ThisDrawing.ModelSpace.AddLine(startPoint, endPoint)

abdisty = abdisty + aboffset

Next count
theform.Hide

End Sub

'padx = theform.TextBox1.Text
'pady = ThisDrawing.Utility.GetReal("Enter Pad Dist. (Y Axis): ")
'platex = ThisDrawing.Utility.GetReal("Enter Plate Dist. (X Axis): ")
'platey = ThisDrawing.Utility.GetReal("Enter Plate Dist. (Y Axis): ")
'abdia = ThisDrawing.Utility.GetReal("Enter Anchor bolt Dia.: ")
'abcount = ThisDrawing.Utility.GetInteger("How Many Anchor Bolts?: ")
'abdistx = ThisDrawing.Utility.GetReal("Anchor Bolt Dist. From Center (X Axis): ")
'abdisty = ThisDrawing.Utility.GetReal("Anchor Bolt Dist. From Plate Edge (Y Axis): ")
Private Sub UserForm_Click()

End Sub
The results of running the base plate routine

- The is the fixblock routine written in VBA, I pasted this particular example because you can see that a few of the routines got much simpler in VBA

Private Sub CommandButton1_Click()
    theform.Hide
    If (ComboBox1.ListIndex = 0) Then
        For Each Block In ThisDrawing.Blocks
            For Each Object In Block
                If (TextBox1.Text <> Null) Then Object.Layer = TextBox1.Text Else:
                    If (TextBox2.Text <> Null) Then Object.color = TextBox2.Text Else: Object.color = 256
            Next Object
        Next Block
    End If
End Sub
ThisDrawing.Regen acActiveViewport

theform.Hide

End If

If (ComboBox1.ListIndex = 1) Then

Dim theSelection As AcadObject
Dim selectedBlock As AcadBlock
Dim thePoint As Variant

ThisDrawing.Utility.GetEntity theSelection, thePoint

If TypeOf theSelection Is AcadBlockReference Then Set selectedBlock = ThisDrawing.Blocks.Item(theSelection.Name)

For Each Object In selectedBlock

If (TextBox1.Text <> Null) Then Object.Layer = TextBox1.Text Else: Object.Layer = 0

If (TextBox2.Text <> Null) Then Object.color = TextBox2.Text Else: Object.color = 256

Next Object

ThisDrawing.Regen acActiveViewport

End If

With ComboBox1

.AddItem (0)

End With

End Sub
The fitblock routine, revised and running in VBA

***for legality sake, I was unable to publish any of the ObjectARX and C++ raw code. Being that VBA is unprotected code, there was no legal issues with publishing it in this document.