Modern Software Quality

with Access and VBA

Philipp Stiefel

VBA – The most dreaded language!

https://insights.stackoverflow.com/survey/2019#most-loved-dreaded-and-wanted

Software Quality?

- ISO 9126 (1991) - superseded by ISO 25010:2011

- Maintainability
  - Analyzability
  - Changeability
  - Stability
  - Testability
  - Maintainability compliance

Immediate effects of **High** Code Quality

- Clear / Without ambiguity
- Easy to understand
- Easy to test
- Easier to change
- Fewer errors

Indirect effects of High Code Quality

- Less development time (long term!)
- Less cost for the client
- More money for your time

Is this relevant for the single developer?

- “Code you have not touched for some time, could as well have been written by somebody else.” – Quote from unknown source.

- Standards help you being accountable!

Coding Conventions

- Option Explicit
- Typed variables
- Code indentation

- Naming conventions?
- Clean Code?
- Commenting code?

Naming conventions

- Hungarian Notation - Leszynski/Reddick

?

Apps Hungarian Notation

```
Dim hwndSomething As Long
Dim cbSomethingElse As Long

hwndSomething = cbSomethingElse
```

Systems Hungarian Notation

```
Dim lngSomething As Long
Dim lngSomethingElse As Long

lngSomething = lngSomethingElse
```

Systems Hungarian + Meaningful Names

```
Dim lngMainWindowHandle As Long
Dim lngBufferSize As Long

lngMainWindowHandle = lngBufferSize
```

Meaningful Names

```
Dim MainWindowHandle As Long
Dim BufferSize As Long

MainWindowHandle = BufferSize
```

Naming conventions

- ~~Hungarian Notation - Leszynski/Reddick~~
- Language
- Standard terminology
- Word separators
- Be careful with acronyms / abbreviations

- **Meaningful and pronounceable names**

Name Things!

- Operation = Name with a method
- Result = Name with a variable
- Constant literal = Name with a constant
- Multiple options = Name with an enum

Unnamed

```
Private Sub ShipOrderBad()
    If DCount("[Invoice ID]", "Invoices", "[Order ID]=" & _
                Nz(Me![Order ID], 0)) > 0 Then
        MsgBoxOKOnly 105
    ElseIf Not IsNull(Me![Shipped Date]) Then
        MsgBoxOKOnly 115
    ElseIf IsNull(Me![Shipper ID]) _
            Or Nz(Me![Ship Name]) = "" _
            Or Nz(Me![Ship Address]) = "" _
            Or Nz(Me![Ship City]) = "" _
            Or Nz(Me![Ship State/Province]) = "" _
            Or Nz(Me![Ship ZIP/Postal Code]) = "" _
    Then
        MsgBoxOKOnly 104
    Else
        Me![Status ID] = 2
        Me![Shipped Date] = Date

        DoCmd.RunCommand &H61
    End If
End Sub
```

Named

```
Private Sub ShipOrderGood()
    If Not OrderIsInvoiced Then
        MsgBoxOKOnly CannotShipNotInvoiced
    ElseIf OrderIsShipped Then
        MsgBoxOKOnly CannotShipOrderShippedAlready
    ElseIf Not ValidateShippingInfo() Then
        MsgBoxOKOnly ShippingNotComplete
    Else
        Me![Status ID] = Shipped_CustomerOrder
        Me![Shipped Date] = Date

        DoCmd.RunCommand acCmdSave
    End If
End Sub
```

Clean Code

"Clean code is simple and direct. Clean code reads like well-written prose. Clean code never obscures the designer's intent but rather is full of crisp abstractions and straightforward lines of control."

- Grady Booch (Object Oriented Analysis and Design with Applications)

Clean Code – Basics

- Follow the standard
- Simpler is always better
- Leave code cleaner than you found it

Clean Code – General code rules

- DRY – Avoid redundancy
- Limit dependencies and coupling
- Depend on abstractions not on concretizations

Clean Code - Methods

- Small
- Do one thing only!
- No side effects
- As few arguments as possible
- No flag arguments to control behavior

Clean Code – Methods – Bad Example

```
Public Function Calculate(ByVal arg1 As Double, ByVal arg2 As Double, ByVal calcType As CalculationType) As Double

    Dim result As Double

    Select Case calcType
        Case CalculationType.Addition
            result = arg1 + arg2
        Case CalculationType.Subtraction
            result = arg1 - arg2
        Case CalculationType.Multiplication
            result = arg1 * arg2
        Case CalculationType.Division
            result = arg1 / arg2
        Case Else
            Err.Raise INVALID_OPERATION, "Calculate", "Invalid CalculationType"
    End Select

    Calculate = result

End Function
```

Clean Code – Methods – Better Example

```
Public Function Add(ByVal addend1 As Double, ByVal addend2 As Double) As Double
    Add = addend1 + addend2
End Function
```

```
Public Function Subtract(ByVal minuend As Double, ByVal subtrahend As Double) As Double
    Subtract = minuend - subtrahend
End Function
```

```
Public Function Multiply(ByVal multiplier As Double, ByVal multiplicant As Double) As Double
    Multiply = multiplier * multiplicant
End Function
```

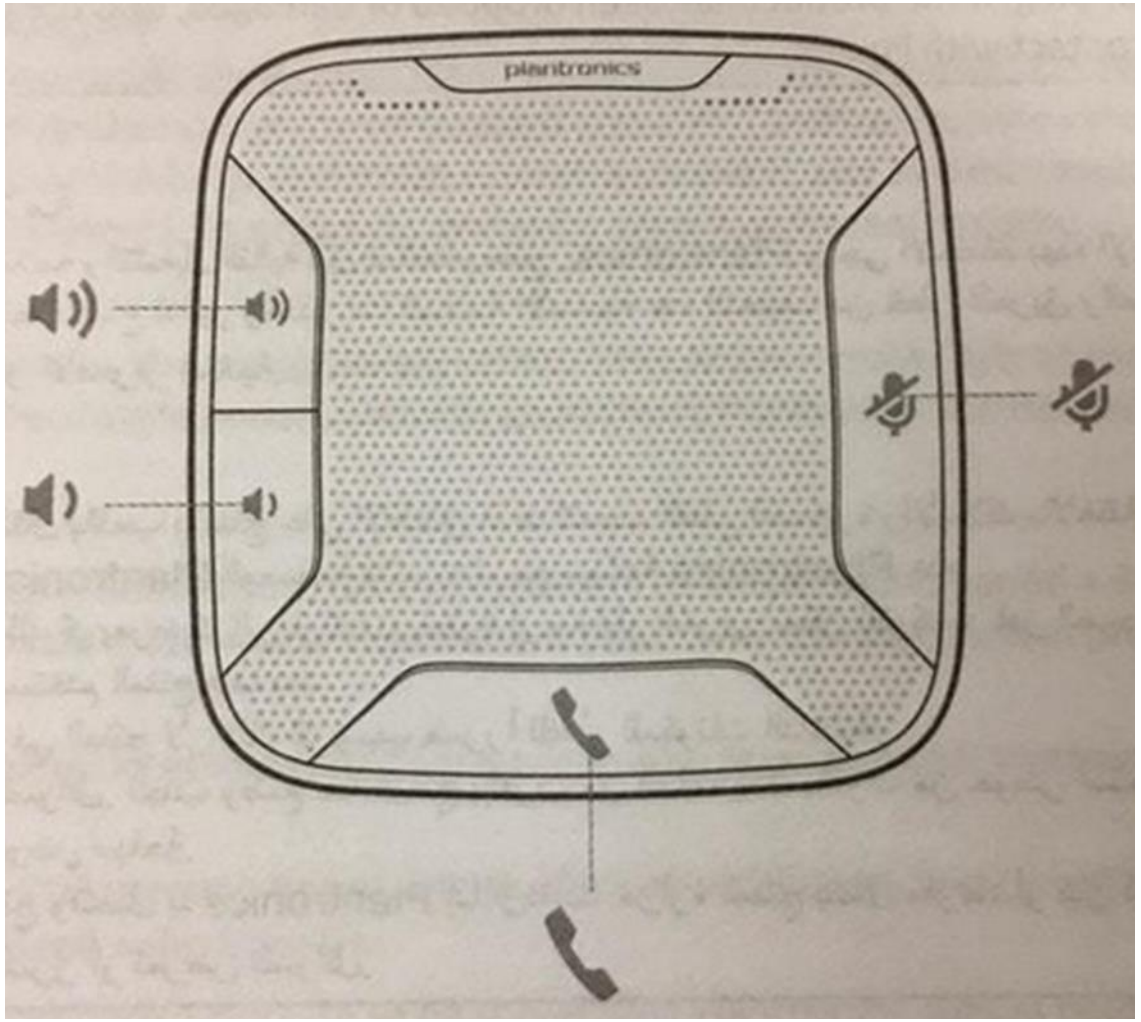
```
Public Function Divide(ByVal dividend As Double, ByVal divisor As Double) As Double
    Divide = dividend / divisor
End Function
```

Clean Code – Classes

- Small
- Do/be one thing only
- Hide internals

- More suggestions for OO-Languages

Comments



Bad Comments – Code Sections

```
'***** Private Methods *****
Public Sub initForm()
On Error GoTo Err
```

```
Public Sub FilterForm(ByVal param As String)
On Error GoTo Err
```

Bad Comments – Explain Parameters

```
' Executes a SQL Command directly
' sp: stored procedure name
Public Function GetCommand(ByRef conn As ADODB.Connection, _
                           sp As String, param As String) As ADODB.Command
On Error GoTo Err

Dim
```

Bad Comments – Procedure Header

```
'---------------------------------------------------------------------------
' Procedure : cmb_Entity_Contact_AfterUpdate
' Author    : [redacted]
' Date      : 01.08.2016
' Purpose   : UPDATE SUB02 Company Data and NACE-Code after update of entity contact
'---------------------------------------------------------------------------
Private Sub cmb_Entity_Contact_AfterUpdate()
    On Error GoTo Err

    m_lngEntity_Contact = Nz(Me!txtEntity_Contact, 0)
    Call Me.Sub02_CompanyData.Form.FilterForm(m_lngEntity_Contact)
    Call Me.Sub02_PersonData.Form.FilterForm(m_lngEntity_Contact)
    Exit Sub
Err:
    Call RuntimeError(Me.Name & ":Cmb_Entity_Contact_AfterUpdate:")
End Sub
```

Commenting Code

- ~~Module and Procedure Headers?~~
  - Out of date
  - Irrelevant information

- ~~Explaining code that is hard to read/understand?~~
  - Make the code easy to read/understand

- Explanation of intent
- Clarification of code

- **Why not What!**

Comment **Why**!

```
' We're Including column headers in the connection and filtering them out in
' the select query for the recordset to force all columns to be of type text.
' Otherwise automatic data type detection will cause errors with misdetected types
oConnection.Open "Provider=Microsoft.ACE.OLEDB.12.0;" & _
    "Data Source=" & m_TargetFolderExcel & _
    ";Extended Properties=""Excel 12.0;HDR=No;ReadOnly=True;IMEX=1"";"

oExcelRecordset.Open "Select * FROM [" & sDataRange & "] WHERE F1 <> '" & FirstColumnHeaderName & "'", _
    oConnection, adOpenStatic, adLockOptimistic, adCmdText
```

Technical Rules

- Complete declarations
  - Methods, Variables, Arguments, …
- No prohibited statements:
  - Stop, End, …?
- No prohibited constructs
  - DoMenuItem, SendKeys, RunCommand, DDE

Tools to check Code Quality

- RubberduckVBA – Inspections
- MZ-Tools – Review Quality
- Total Access Analyzer

Demo – RubberduckVBA + MZ-Tools

- RubberduckVBA – Inspections
- MZ-Tools – Review Quality
  - MZ-Tools Settings and Rules

Automatic Testing

Unit Test / TDD – Quick Summary

- Unit Tests
  - Verifying functionality automatically
  - Fast
  - Isolated
- Test Driven Development – TDD
  - Test First – Functionality second
  - Quick feedback loop
  - Better design

Unit Tests in Access / VBA

- Testing is hard with Access
  - Databases hard to test
  - UI hard to test
  - Access glues data and UI together
- Easy to test
  - Functions (without UI/data access)
  - Classes
- **At least: Keep test code you write anyway!**

Demo – Unit Test

- … with RubberduckVBA
- … with AccessUnit fork

Units Tests vs. Integration Test

- Unit Test
  - Isolated
  - No dependencies
- Integration Test
  - Multiple units combined

Tools for Testing Code

- RubberduckVBA – Unit Tests
- AccUnit
  - Problem: Dependency on obsolete Software (SimplyVBUnit 3.0)
- AccessUnit
- AccessUnitFork

UI Automation Tests

- Very time consuming to create and maintain
- Error prone during execution
- Very difficult to implement for Access

Source code control

- Separates metadata and code
- History – Tells you “why“ and “what”
- The “Single source of truth“
- Base for automated processes

Continuous Integration / Delivery (CI/CD)

- General Software
  - Build servers
- Access specific
  - Build scripts

Automating Code-Quality Checks

- MZ-Tools Automation features

  https://www.mztools.com/v8/onlinehelp/MZTools8Help.html?automating_features.htm

Automating (Unit-) Tests

- AccUnit – Should be possible - hardly documented
- RubberduckVBA – No automation
- AccessUnit – No automation
- AccessUnit-Fork – Work in progress

What about (T)SQL?

- Functions degrade performance
  - Generally
  - Especially when preventing SARGability
- Formatting even more important
- Comments may be more useful
- No comments in Access SQL

You are professionals!

Care about the quality of your code!

Thank You!

Let’s connect

linkedin.com/in/philipp-stiefel

@philivc

https://codekabinett.com/en

Reading List - Books

- Clean Code: A Handbook of Agile Software Craftsmanship - Robert C. Martin
- Working Effectively with Legacy Code - Michael Feathers
- Test Driven Development: By Example – Kent Beck
- The Art of Readable Code – Dustin Boswell / Trevor Foucher

(Links on this slide are Amazon Affiliate Links)

References & Links

- MZ-Tools – Excellent multi purpose developer tool for VBA
- AccUnit – Unit Testing framework for Access/VBA
- RubberduckVBA – Code Inspections, Unit Tests and much more
- accessUnit (original) – Unit Test framework for (and written in!) Access/VBA
- accessUnit-Fork – My adaption of accessUnit also usable as Add-In