What makes a good Visual Basic .NET program(mer)?

Our real aim

- Confident and aggressive so that you learn from the experience
- Rather than, desperate, superstitious and accepting of ‘any’ solution that works.

Expert programmers

- Only got to be an expert by making lots and lots of mistakes
  - Experimentation is very important
  - Would have learned from others
- May not be a good programmer at all!

What makes a good program?

- Working in a predictable reliable manner and to specification isn’t enough!
- Interface considerations of course
  - Why
- Can it be modified and maintained?
  - Why

The role of programming standards

- Absolutely vital when more than one programmer is involved
  - That is always if the program is useful
- Help to remove the ‘personality’ and ‘ego’ from the code

Extension for Assignment 4 and Exam Info

- Extension!!
  - Working system can now be handed in in week 13, but...
  - Presentations will happen in week 12
- Exam Interviews
  - Will happen in week 13
  - Book times in week 12
  - 15 minutes each
  > Reflective style questions
  > May be on anything covered during the semester
  - Will involve studio academic and tutor if scheduled during the studio.

Our real aim

- Confident and aggressive so that you learn from the experience
- Rather than, desperate, superstitious and accepting of ‘any’ solution that works.

Expert programmers

- Only got to be an expert by making lots and lots of mistakes
  - Experimentation is very important
  - Would have learned from others
- May not be a good programmer at all!

What makes a good program?

- Working in a predictable reliable manner and to specification isn’t enough!
- Interface considerations of course
  - Why
- Can it be modified and maintained?
  - Why

The role of programming standards

- Absolutely vital when more than one programmer is involved
  - That is always if the program is useful
- Help to remove the ‘personality’ and ‘ego’ from the code
What should they cover?

- **Design**
  - Modules, procedures, components
- **Naming conventions**
- **‘Look’ of the code**
- **Interface**
  - User interface should be analyst’s problem but still programmer’s make some decisions
- **Version control**

Formatting code

- **Why does it matter?**
  - Make code easier to read and understand by visual organisation
  - Make the code self documenting
  - Reducing the effort requirement to understand code
  - Helping the reader by not forcing them to make assumptions about your code
- **All from Foxall (2000)**

Formatting standards

- Don’t place more than one statement on one line
- Use the line continuation character
- Use indentation to show structure
- Use white space to group related statements

Don’t place more than one statement on one line

**This:**

```plaintext
Aspect = Abs(intYLeg / intXLeg): xCenter = m_rectBound.Left + (m_rectBound.Right - m_rectBound.Left) / 2: yCenter = m_rectBound.Top + (m_rectBound.Bottom - m_rectBound.Top) / 2
```

**Or this:**

```plaintext
Aspect = Abs(intYLeg / intXLeg)
xCenter = m_rectBound.Left + (m_rectBound.Right - m_rectBound.Left) / 2
yCenter = m_rectBound.Top + (m_rectBound.Bottom - m_rectBound.Top) / 2
```

Use the line continuation character

- Previous e.g. is a good one
- Never let a line be longer than 90 characters
- Use it in a sensible place
- Also use it correctly!
  ```plaintext
  strMessage = “Hello there this is a very long string _ that goes over the line that I am typing into.”
  strMessage = “Hello there this is a very long ” & _
  “string that goes over the line that I am typing
  ```

Use indentation to show structure

```plaintext
If strText = “” then
  NoZeroLengthString = null
Else
  NoZeroLengthString = strText
End if
```

```plaintext
If strText = “” then
  NoZeroLengthString = null
Else
  NoZeroLengthString = strText
End if
```
Use white space to group related statements

- Insert a blank line
  - Before and after each if...then
  - Before and after each Select Case
  - Before and after each loop
  - After declaring a block of variables
  - Between groups of statements that perform unified tasks

Why comment

- Feels like a waste of time when time is tight
  - Lots of comments aren’t all that helpful
- Try to,
  - Document the purpose of code (why not how)
  - The thinking and logic behind the code
  - Call attention to important turning points
  - Reduce the need for users to ‘run’ the code in their heads

Too often comments are like this

```
" intAge > c_LegalAge then process
" the sale
If inAge >= c_LegalAge then
  call ProcessSale
End if
```

This is better

```
" if the buyer is of legal age
" to purchase alcohol, process
" the sale
If inAge >= c_LegalAge then
  call ProcessSale
End if
```

Other comments

- If you need to violate good design, explain why
- Document when an error is expected and why
- Comment before writing the code!
- Make your comments readable
- Give each procedure a comment header

My pet hate - comment boxes

- **********************************
- * Don’t ever do this they are *
- * big pain to maintain       *
- **********************************
Version control

- Absolutely vital when you have a team
- Goals are to
  - Ensure that the most current and stable version is the one compiled
  - Maintain a history of all changes
  - Maintain a backup of all source files to prevent loss of work

Some tips

- Display version number in the About box
- Increment the number each time you compile
- Document changes in a “readme” file
- Backup your files carefully
- Try using the .NET source control tool
  - If you’re brave

Yes, readme’s can be good

Look for these is remainder bookshops

Reminders

- Reminder about coding standards
  - You will be submitting your code and it will be marked on the basis of code comments and structure.
- Reflective Journals
  - All journals *must* be submitted. We will soon be doing an audit, and if you haven’t submitted your journals, you will not pass the hurdle requirement
  - There is a rumour circulating that you can submit 50% of your journals and pass. This is not the case.