Software Lessons Learned

How to write Absolutely Terrible Software!

How to write Absolutely Terrible Software!

- Absolutely Terrible Software (ATS) requires at least one of the following:
  - Terrible Comments,
  - Terrible Headers, or a
  - Terrible Coding Style

Terrible Comments

- The terrible approaches to comments in source code are:
  - TAKE NO HOSTAGES

    the He-Man comment approach
  - CONFUSE THEM WITH FACTS

    the passive aggressive comment approach
  - HE WENT THAT AWAY ←→

    the job security comment approach

TAKE NO HOSTAGES
(or the He-Man comment approach)

- Who needs comments?
  - if you cannot understand my code without comments then you have no business calling yourself a programmer.

CONFUSE THEM WITH FACTS
(or the passive aggressive comment approach)

- I added comments because the boss wants them...
  - but I am not going to waste my time by telling you anything useful…
    - So I'll repeat each line of code with a comment that says the same thing…

```
I := I + 1;    // add one to I.
```

HE WENT THAT AWAY ←→
(or the job security comment approach)

- I added comments because the boss wants them...
  - but its my code…
    - And to make certain it stays that way, my comments are misleading, encrypted, or irrelevant!

Comments: The Better Way

- five kinds of comments:
  - repeat of the code ***AVOID***
  - explanation of the code ***AVOID***
  - marker in the code ***AVOID***
  - summary of the code ***BETTER***
    - distills a few lines of code into 1 or 2 sentences
  - description of the code's intent ***BEST***
    - explains the purpose of a section of code

Terrible Headers

- The terrible approaches to file and subroutine headers are:
  - IT IS TOO OBVIOUS TO BE NEEDED
  - EVERYTHING BUT THE KITCHEN SINK

IT IS TOO OBVIOUS TO BE NEEDED

- Well, it is perfectly obvious to the casual observer that the third parameter is the z-axis velocity in meters per hour…

```
SUBROUTINE Comp(x0,x1,x2:FLOAT);
```

EVERYTHING BUT THE KITCHEN SINK

```
{}FUNCTION AppendChar(aStr:STRING; aChr:CHAR):STRING;
{+-------------------------------------------------------------------------\
 | Name:      AppendChar                                                    \
 |                                                                           \
 | Purpose:   Append the specified character to the end of the string and |
 |            return the result as the function value.                     |
 |                                                                         |
 | Algorithm: If the length of the passed string is less than the maximum |
 |            string length, the passed character is appended to the end  |
 |            of the string and the result is returned as the function    |
 |            value.                                                      |
 |                                                                         |
 | Inputs:    aStr  -- the string to which the character is to be appended|
 |            aChr  -- the character to append to the string aStr.        |
 |                                                                         |
 | Outputs:   The function value is the appended string                    |
 |                                                                         |
 | Called by: CreateBorder, DoReport, EndPage, ForSummary, PrintHeader    |
 |                                                                         |
 | Author:    John P. Verbose                                              |
 |                                                                         |
 | Created:   2/1/86                                                       |
 |                                                                         |
 | Revised:   *                                                            |
 +-------------------------------------------------------------------------+}
  BEGIN
    AppendChar := aStr + aChar;
{}END {AppendChar};
```

EVERYTHING BUT THE KITCHEN SINK (2)

- The vertical bars at either end are difficult to maintain
- The *called by* section is difficult to maintain
- The complete boilerplate is overkill for such a trivial function
- The algorithm section is strained.
  - it is hard to describe something as simple as adding a character to the end of a string at a level of detail that is simpler than the code itself

Headers: The Better Way

- For files:
  - Statement of purpose
  - who wrote it, when
  - (optional) list of references (name the book or magazine article that describes the algorithm or algorithm being implemented)
  - (optional) limitations or cautions that may not be otherwise readily apparent
  - date and general description of revisions

Headers: The Better Way (2)

- For subroutines:
  - Statement of purpose
  - parameter descriptions
    - *what it is
    - *what it does
    - *what's its valid range
    - *is it an input, an output or both
    - *what's its units (is it speed in furlongs per fortnight?)
  - who wrote it, when
  - date and general description of revisions

Comments & Headers: The Better Way

- Use comment & header styles that don't break down or discourage modification
- Comment as you go along
- Avoid self-indulgent comments
- Write comments at the level of the code's intent

Terrible Coding Style

- The terrible approaches to coding style:
  - IF IT TOOK ME 3 WEEKS TO WRITE IT SHOULD TAKE YOU 3 WEEKS TO FIGURE WHAT I DID

IF IT TOOK ME 3 WEEKS TO WRITE IT SHOULD TAKE YOU 3 WEEKS TO FIGURE WHAT I DID

- Terse code is “good”
- The best variable names are I, J, X, Y and Z
- Indented code wastes space characters
- Blank lines are a precious resource - don’t waste them either
- I wrote it, so it’s perfect

Coding Style: The Better Way

- Use intuitive and descriptive subroutine, constant and variable names.
  - Use naming conventions to identify scope & type
- Indent the body of code blocks
- Use blank lines to separate logical blocks.
- Separate subroutines by 3-5 blank lines
- Read your code
  - Read it out-loud or have other people read it.