**Chapter 32: Self-Documenting Code**

* Most programmers enjoy writing documentation if the standards aren’t unreasonable
  + It’s a symbol of professional pride
* “Code as if whoever maintains your program is a violent psychopath who knows where you live”

**External Documentation**

* Unit development folders
  + Informal document that contains notes used by the developer during construction
  + Typically a unit is like a class
  + Point of the UDF is to maintain a trail of decisions made during development
* Detailed Design Document
  + Describes class level or routine level design decisions
  + Also contains alternatives considered
    - And reasons for selecting the implemented solution

**To Comment or Not to Comment**

* Comments are easier to write poorly than well
* Sometimes commenting can be more damaging than helpful

Kinds of Comments

* Repeat of the Code
  + Not useful
* Explanation of the Code
  + Usually better to make the code less complicated than try to explain it
* Marker in the Code
  + Developer note to finish up or do something
* Summary of the Code
  + Distills a few lines of code into one or two sentences
  + Nice because can scan more easily than the code
  + Useful
* Description of the code intent
  + Useful if short
  + **Focus on the why, not the how**
  + Meh:
    - Find ‘$’ in inputString
  + Intent
    - Find command-word terminator ($)
* Information That Cannot Possibly Be Expressed by the Code Itself
  + Useful and necessary for random stuff

Commenting Efficiently

* Pick a style and stick with it
* Make sure the style isn’t hard to maintain
* The longer the comment, the more annoying, less useful and harder to maintain
* Use the Pseudocode Programming Process to reduce commenting time
* Make every comment count
* Document surprises
  + Not generic knowledge
* Differentiate between major and minor comments
  + Underline or something
* Comment anything that gets around an error
* Justify violations of good programming style

Commenting Data Declarations

* Comment units
* Comment ranges of allowable values
* Comment what coded values mean (map to)
* Comment limitations of input data
* Document global data

Commenting Control Structures

* Provide reason for decision and summary of the outcome
* Put a comment before each if, loop and block of statements

Commenting Routines

* Many textbooks urge you to stack information at the top of every routine, regardless of its size or complexity
* This is bad b/c overhead is so high people will avoid making new routines to avoid doing commenting overhead
* Guidelines
  + Keep comments close to the code they describe
  + Describe each routine in one or two sentences at the top of the routine
    - You shouldn’t have to think hard about this
  + Differentiate between input and output data
  + Document interface assumptions
  + Maybe comment on the routines limitations
  + Document source of algorithms that are used

Commenting Classes, Files, and Programs

* General Guidelines for Class Documentation
  + Describe the design approach to the class
  + Describe limitations, usage assumptions and so on
  + Comment on the class interface
* General Guidelines for File Documentation
  + Describe the purpose and contents of each file
  + If the file contains more than one class, explain why the classes needed to be combined into a single file
  + Put your name and contact tin the comment block so folks can contact you
    - Authorship and ownership is good
  + Include a version control tag
  + Include legal notices
  + Give the file a name related to its contents