The Seven Deadly Sins of Programming

I. Not writing an algorithm or not testing an algorithm after it is written.

Skipping the algorithm phase of programming is a costly error which can lead to hours of fruitless "hacking". *Even after an algorithm is written,* it must be tested by "hand tracing through" the algorithm. If such testing is not done, the programmer will be wasting their time during the translation phase of the programming process.

II. Trying to write an algorithm before the problem that needs to be solved is understood.

Later when the programmer gets a clearer idea of what the problem is all about, they will have to make serious changes in their algorithm. This can lead to a series of ill-conceived "patches" to the old algorithm, which will waste a lot of time.

III. Trying to correct an errant program before understanding why it doesn't work.

It is truly astounding that some programmers believe that they can fix a bug without understanding what causes the bug! The pattern of self-destructive behavior runs something like this: A bug is discovered and rather than take the time to understand the bug, the programmer "tries" various fixes. When following this route the programmer often reaches a point of desperation, even taking advice from people who haven't even looked at their program!

IV. Changing an algorithm in order to fix a syntax error.

This is a variation of deadly sin III. The programmer has a syntax error, but can't seem to fix it. Instead of doing the necessary research on the statement causing the syntax error, the programmer uses a different algorithm!

V. Writing a program without subprograms and then trying to subdivide it later.

Rather than carefully plan the appropriate subprograms, some programmers write an overly long main program, then subdivide the code later. *This is guaranteed to waste time* and throws away the "divide and conquer" benefits of subdividing the program at the beginning.

VI. Attempting to write and debug the entire program, rather then finish one subprogram at a time.

The main program and subprogram stubs should be done first. A program stub is a function that has a header and delimiting braces, but no code. After this code successfully compiles, the subprogram bodies can be filled in one at a time. *When possible, these subprograms should be debugged separately,* using a test program designed to test the subprogram. After they are known to be correct, they can be added to the main program.

VII. Not learning to use the debugger.

It's hard to understand why one would not learn to use the debugger! It takes about thirty minutes to get comfortable with a very useful subset of the debugger commands. Programmers who commit any of the seven deadly sins often *spend hours* using ineffective debugging techniques every time they write a program.