Forth Methodology Applied to Programming:

1. **BEGIN BEGIN BEGIN**
   Identify and reject your illusions and ideas that don't help.

2. Identify and reject the non-problems blocking your problem or
   keep them but trick them into helping solve the real problem.

3. First consider the options that everyone else would reject first,
   that is where the biggest algorithmic improvement is likely to be found.

3. Carefully construct a well thought out solution.

4. Optimize that solution design as far as it can go in that direction.

5. Identify and reject the illusion that starting over is bad and you
   are ready to code. Return to 1. with a greater understanding of
   the issues **UNTIL** you collect all the good solutions that
   you can imagine.

6. Compare solutions. Construct experiments, benchmarks, simulations
   or whatever you need to confirm that you found the optimal
   approach to the problem.

7. Continue to return to 1. **UNTIL** the ideas are "right by design."

8. After making the problem look brutally simple by doing the
   factoring factoring factoring before coding approach the
   trival programming task of constructing a one-to-one
   image of the optimal solution in code.

9. Code.
   Build custom tools if they help.
   Write code so simple and clear that bugs simply can't happen.
   Make the code "right by design."
   :define using one-liners about this long ;
   Interactively test each Forth word.
   Extend the core language making your custom language and moving
   you toward your solution.
   Return to 1. **UNTIL** the code solution falls out.

10. Write documentation so simple and clear that bugs simply can't happen.
   Document the "right by design" algorithms and code.
   Create a glossary with a description of each word.
   Have fun at all times.
   Enjoy the satisfaction of a job well done.

If you do this right the easiest step is 9.
If you start at 9 you understand 1% of this methodology.

This methodology, framing a programming task here, can be applied
to designing hardware, it has, or to any type of problem.

The inventor of Forth, Charles Moore, says that his language was designed
to "avoid" the "unsolveable" problems in computer science.
He says that the "real" problem is the problem that you

want to solve. Adding unsolvable problems to the real problem is really your problem.

On each iteration you understand the problem better, each time you go through the loop, you find remaining places for new optimization. After you found and removed the last bottleneck you find the next largest bottleneck in each sucessive iteration.

Mr. Moore repeated steps 1-10 on software for 20 years. He declared that the programming problem had been solved. The remaining problem was hardware. Mr. Moore studied the problem and constructed his own new tools.

Mr. Moore repeated steps 1-10 on hardware/software for another 20 years. Computers are hardware/software and Mr. Moore has been getting closer.

He used step by step iterative changes. He said that he made a couple of wrong turns on a couple of landings and had to go back to find the up staircase again. Learn to recognize when you have left the up staircase. I admire him for all the times when he told us that he had been wrong in an approach or in a conlusion, that he had taken the wrong path before or told us something that was not true.

I see a lot of people taking a leap, or climbing up the vertical wall. I tell them that there are stairs and that it is a longer, but easier and better path. But some people will get stuck trying to go up the down escalator. Consider using the up stairway.

If you find an up escalator somewhere please let us all know!

Jeff Fox 12/09/01